CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Experimental analysis of RTEMS in multi-core platforms

Rúben Gonçalves
Joel Pinto

CISTER-TR-181008

Experimental analysis of RTEMS in multi-core platforms

Rúben Gonçalves, Joel Pinto

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

In recent years, the demand for the use of embedded multiprocessor systems in everyday products and in critical systems has grown exponentially, forcing the real-time community to follow this trend. For this, the development and adaptation of RTOS (Real-Time Operating System) to multiprocessor platforms became critical. Meanwhile, a new case of multiprocessor platforms appeared, SMP (Symmetric Multiprocessor), a case that affected the community and make most of the RTOS to be adapted to this new kind of platform.RTEMS (Real-Time Executive for Multiprocessor Systems), a free open source real-time operating system designed to support embedded applications with the most stringent real-time requirements while being compatible with open standards such as POSIX, was one of those RTOS who was recently adapted to SMP platforms by the community.Unfortunately, this adaptation is still not perfect, and for the work already done, a lot of testing must be performed, to verify the correct implementation and behaviour of the protocols and mechanisms that had been adapted to Symmetric Multiprocessing.So, the aim is therefore to do investigation on RTEMS, making use of QEMU to emulate a SMP platform. On an earlier phase, this work contemplates studying and understanding the innerworkings of RTEMS, followed with the creation of a new feature to help on the understanding of the operating system, the development of samples tests (RTEMS applications) to test the directives implemented for SMP environments, and finally the implementation of a famous case study.

Experimental analysis of RTEMS in multi-core platforms

2017/2018

1150785 – Ruben Filipe Gonçalves

1150963 – Joel Oliveira Pinto

Experimental analysis of RTEMS in multi-core platforms

Cister

2017/2018

1150785 – Ruben Filipe Gonçalves

1150963 – Joel Oliveira Pinto

Bachelor's Degree in Computer Engineering

ISEP Supervisor:

- Luis Miguel Nogueira

External Supervisor:

- Luis Miguel Pinho

Acknowledgments

We are grateful to CISTER for this amazing opportunity, especially to Luis Miguel Pinho and Luis Miguel Nogueira who proposed the realization of this project under PESTI, and Claudio Maia who helped us in almost every step of the project.

Joel and Ruben

A special recognition to everyone who helped me during this last three years, family and friends.

Joel Pinto

I want to take this opportunity, to thank my parents and grandmother for always showing me what real work looks like. Thank you to my friends and girlfriend for being true to me and for knowing how to have a good time.

Ruben Gonçalves

Abstract

In recent years, the demand for the use of embedded multiprocessor systems in everyday products and in critical systems has grown exponentially, forcing the real-time community to follow this trend. For this, the development and adaptation of RTOS (Real-Time Operating System) to multiprocessor platforms became critical. Meanwhile, a new case of multiprocessor platforms appeared, SMP (Symmetric Multiprocessor), a case that affected the community and make most of the RTOS to be adapted to this new kind of platform.

RTEMS (Real-Time Executive for Multiprocessor Systems), a free open source real-time operating system designed to support embedded applications with the most stringent real-time requirements while being compatible with open standards such as POSIX, was one of those RTOS who was recently adapted to SMP platforms by the community.

Unfortunately, this adaptation is still not perfect, and for the work already done, a lot of testing must be performed, to verify the correct implementation and behaviour of the protocols and mechanisms that had been adapted to Symmetric Multiprocessing.

So, the aim is therefore to do investigation on RTEMS, making use of QEMU to emulate a SMP platform. On an earlier phase, this work contemplates studying and understanding the innerworkings of RTEMS, followed with the creation of a new feature to help on the understanding of the operating system, the development of samples tests (RTEMS applications) to test the directives implemented for SMP environments, and finally the implementation of a famous case study.

Keywords (Theme): Embedded systems, Real-Time systems, Real-time operating systems.

Keywords (Technologies): C, RTEMS, QEMU.

Resumo

Nos últimos anos, o uso de sistemas embebidos em plataformas multiprocessadores para produtos do dia a dia e sistemas críticos tem vindo a crescer exponencialmente forçando a comunidade ligada aos sistemas de tempo real a seguir essa tendência. Para isso, o desenvolvimento e adaptação de sistemas operativos de tempo real (Real Time Operating System) para plataformas de multiprocessador tornou-se crítico.

Enquanto isso, surgiu um novo caso de plataformas multiprocessador, o multiprocessamento simétrico (Symmetric Multiprocessing), um caso específico de uma abordagem que pode ser tomada para multiprocessadores e que representa o principal foco de adaptação dos RTOS a esta plataforma.

O RTEMS (Real Time Executive for Multiprocessor Systems), um sistema operativo de tempo real, de código fonte aberto, projetado para suportar aplicações embarcadas com os requisitos mais rigorosos em tempo real e compatível com padrões abertos como o POSIX, foi um desses RTOS que foi recentemente adaptado para Plataformas SMP pela comunidade.

Infelizmente, esta adaptação encontra-se longe da perfeição, e para o trabalho já realizado, muitos testes devem ser realizados, para verificar a correta implementação e comportamento dos protocolos e mecanismos que foram adaptados para o multiprocessamento simétrico.

Assim, o objetivo é, portanto, fazer uma investigação sobre o RTEMS, fazendo uso do QEMU para emular uma plataforma SMP. Numa fase inicial, este trabalho contempla o estudo e compreensão do funcionamento interno do RTEMS, seguido da criação de uma nova funcionalidade para ao seu estudo, o desenvolvimento de test-suites (aplicações RTEMS) para testar as diretivas implementadas para funcionar em ambientes SMP e, finalmente, a implementação de um famoso caso de uso.

Keywords (Theme): Sistemas embebidos, Sistemas de tempo real, Sistema operativo de tempo real.

Keywords (Technologies): C, RTEMS, QEMU.

Glossary

This Section shows all the referenced concepts, symbols and acronyms.

Expression	Meaning
CISTER	Research Centre in Real-Time and Embedded Computing Systems
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating Systems
POSIX	Portable Operating System Interface
SMP	Symmetric Multiprocessing
AMP	Asymmetric Multiprocessing
CPS	Cyber Physical Systems
CPU	Central Processing Unit
BSP	Board Support Package
EDF	Earliest Deadline First
DP	Deterministic priority
ISR	Interrupt Service Routine
SP	Simple Priority
FP	Fixed Priority
ASR	Asynchronous Signal Routine
APA	Arbitrary Processor Affinity
NOP	No Operation
MrsP	Multiprocessor Resource Sharing Protocol
OMiP	O(m) Independence Preserving Protocol

Contents

Index of tables

Index of figures

1. Introduction

This chapter begins by presenting the project and the reason for its development, giving also an insight of its work field, real-time and embedded systems.

1.1 Project Context

With the aim of fulfilling the development of the students on a less academic context, ISEP (Instituto Superior de Engenharia do Porto) presents the opportunity to realize an internship in the third and last year of Computer Engineering bachelor's degree for the curricular unit of PESTI. This curricular unit aims to apply the knowledge learned during the bachelor's degree, as well as personal, interpersonal and social skills, for the design of engineering solutions.

The internship was developed by a team of two students, achieved in cooperation with the Research Centre in Real/Time & Embedded Computing Systems (CISTER) and focused on the research area of Real-Time and Embedded Systems.

This research field is responsible for studying real-time systems. A real time system is a system that is developed and analyzed to guarantee a worst-case response time to critical events, as well as acceptable average-case response time to noncritical events [1]. To control those devices, Real-Time Operating Systems (RTOS) may be used. The difference between an Operating System and a RTOS lays in the nature in how they approach each task. Standard operating system focus on doing as much computation in the shortest span of time, while RTOS emphasize on having a predictable response time, offering accuracy to real-time events, allowing a higher deterministic reaction to external events [2].

A multitude of everyday products use computing devices, making the demand for those devices to grow exponentially and always requiring more and more performance. With this, the industry tried to develop better processor chips, but they faced hardware physical limitations, since the increase in frequency reached thermal/energy bounds. So, the transition to multiprocessor systems to amplify computing times and overturn the physical limitations seemed as the next logical step [3].

A special case of multiprocessor systems is the SMP, Symmetric Multiprocessing, a computing architecture where two or more processors are attached by the same

access mechanism to a single, shared memory and controlled by a single operating system instance. The processors share the memory device through a common high-speed bus and have to contend to access it.

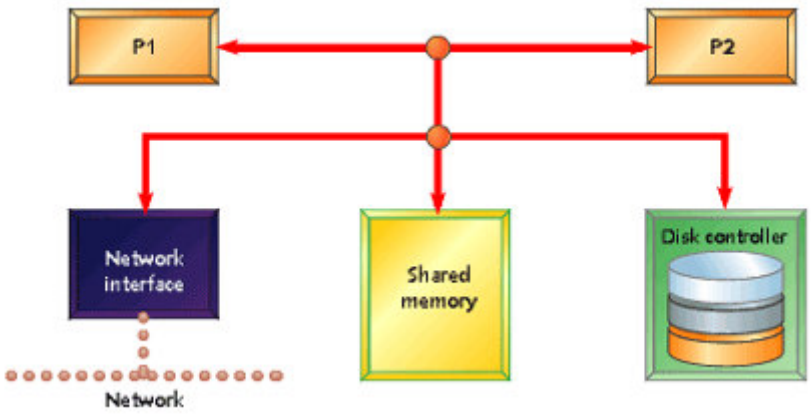


Figure 1 Symmetric Multiprocessor architecture
data source: http://www.rtcmagazine.com/express-logic/

As the real-time systems community started to slowly integrate symmetric multiprocessing systems in their products, many problems appeared, namely, the grown of the synchronization overhead, expensive costs of thread migration, (i.e. threads moving from and to other cores (processor)), the technical troubles for the development of appropriate locking protocols and scheduling mechanisms for multiprocessor without an exponential gain of overhead, hence an extremely slow and disquieting transition [4].

Despite the slow change, multiple Real-Time Operating Systems already offer Symmetric Multiprocessing support, though some of them do not have the expected and correct behaviour regarding some mechanisms, therefore they must be tightly studied and tested.

1.2 Project Presentation

Following the context presented on subsection 1.1, this project had as main goal to study and test RTEMS (Real-Time Executive for Multiprocessor Systems) on a SMP platform, an ARMV7-A architecture. An extensive, methodological study of RTEMS was imposed to fully understand it.

The team focused on two critical mechanisms that bring an enormous amount of trouble when being developed to SMP architectures, the scheduling and synchronization/communication mechanisms.

The scheduling mechanisms can provide immediate response to specific external events, particularly the necessity of scheduling tasks, this means assign resources that will be able to complete the work, doing it within a specified time limit after the appearance of such tasks.

Besides this, the synchronization mechanisms are one of the most critical, since a fundamental part of the work that OS practises is controlling which task accesses what, at which time. Synchronization also allows to modify the scheduling when the control is taken away from a process when it was not necessarily prepared to give up control.

To acquire a higher comprehension of the operating system, samples were developed, those samples helped to fully understand the behaviour of the mechanisms. Also, to verify the correct behaviour, a buffer was implemented in RTEMS kernel, it registered all the preemptions in all CPUs, this is, a higher priority task gets control of the CPU and the lower priority task will wait again (goes back to the ready queue),

And finally, with an integral awareness of the operating system, a real time case study was suggested to be implemented on top of the RTEMS OS, allowing the team to evaluate their knowledge and to go through the adversity of developing a real time application for SMP.

1.3 Organization presentation

CISTER (Research Centre in Real-Time and Embedded Computing Systems) [5] is a research unit created in 1997 based at the School of Engineering (ISEP) of the Polytechnic Institute of Porto.

Since the beginning, CISTER has grown to become one of the best European research units, it has well-established roots in Real-Time and Embedded Computing Systems scientific community and works in a number of subjects such as:

1. Real-Time communication networks and protocols.
2. Distributed embedded systems.
3. Wireless sensor networks (WSN).
4. Cyber-Physical systems (CPS).
5. Real-Time programming paradigms and operating systems.
6. Scheduling and schedulability analysis.
7. Cooperative computing and QoS-aware applications.

As its strategic vision, the unit has been consistently able to identify and contribute to emerging topic in the area and continues to do so with a strong tradition of developing foundational work with a vision for the future in areas such as:

- Next generation of computing systems programming paradigms.
- Modelling and analysing temporal behaviour.
- Handling the requirements of mixed-criticalities.
- Resource management in energy-aware computation.
- Real-Time communication protocols that provides mobility, ubiquity and pervasiveness.
- New demands at all layers of complex systems for better resource QoS management.

It is important to highlight that CISTER was, in 2004 and 2007 awarded with the classification of Excellent in the FCT evaluations.

1.4 Document organization

This report is divided into eight main chapters, Context, Work Environment, RTEMS explanation, the integration of the operating system, Samples description, the circular buffer, case study and for last, conclusions.

Chapter 2, Context. In this case we start with the description of the problem and elaborate onto state of the art and business opportunities.

Chapter 3, Working Environment, explains the methodologies and technologies used in this project, also showing the planning of the project and the meetings to demonstrate the evolution of the project during his lifetime.

Chapter 4, RTEMS, this chapter explains all the pertinent point of the real-time operating system.

Chapter 5, Integration on an emulator, this section will explain how the hypervisor QEMU supports and boots the operating system RTEMS.

Chapter 6, Circular buffer implementation, contains the analysis, design, solution and results of the kernel buffer.

Chapter 7, Scheduling and Synchronization Samples, starts by explain what has been tested and the purpose, then presenting the analysis, implementation and results of the developed samples.

Chapter 8, Mine Control case study, presents the case study implemented and all the related information,

Chapter 9, Conclusions, summarize the conclusions regarding all aspects of the project. In a first section, it recaps the work done, enumerating the strengths and positive side of the work developed and finishing with the future work.

2. Context

On this chapter, the problem of incorporating multi-core processors in safety-critical systems is presented, followed by an exposition on how the very different parties involved have been dealing with this process and what is the direction followed. Ending the chapter with the business opportunities and what benefits from these studies.

The Context is thereby split in 3 sub-sections:

Sub-section 2.1 The Problem, where we address the specific tasks this work looks to tackle. This chapter is subdivided in 3 sub-sub-chapters. These enter in detail on the problems introduced right before.

Sub-section 2.2 State of the art, where we introduce the state of development of RTOS to support SMP environments. Following an analysis on the state of the art of RTEMS.

Sub-section 2.3 Business Areas/Opportunities introduces the reader to cyber-physical systems and explains the opportunities these devices have on current industries.

2.1 The problem

Even though the concept of a multiprocessor system has been around for decades, only recently it attained commercial viability as demand for more resources and processing power grows.

The introduction of SMP platforms brought fundamental changes from the uniprocessor environments, specifically the scheduling and synchronization processes. It appeared a new dimension on the scheduling process. Now the scheduler, besides having to choose which tasks to run, it has to decide where to run them, while still maintaining the efficiency requirements the same. Adding to this, the resource sharing happening between tasks gain a, perhaps even critical role due to the appearance of true task concurrency.

The real-time operating system RTEMS support for multiprocessing is a very recent one, as available processor platforms for real-time systems have been single-core only, up to very recently. This means the solutions found and implemented to solve various problems can still be not very well documented or even present some unexpected behaviours. By being an open-source RTOS, the RTEMS Projects encourages developers to help and report bugs or different problems that they might come across.

The lack of test suites targeting the scheduler algorithms implemented and the synchronization and messaging managers of the RTOS seemed therefore a pertinent problem to tackle, moreover, the team saw the fitness to develop a feature that would allow them to check the entrance and leaving threads from the processors, to compare with the established expected results of the new samples.

Since the leap between the development of uniprocessor-to-multiprocessor-based architectures, and even the development of operating systems to run on them, is not as trivial as some would think, the emergence of multiprocessor solutions in critical real-time systems is yet in an embryonic state. This fermented the suggestion to implement the rather famous, in the academic context, “mine control system” case study [6] that came to be as a mean of showing a multiprocessor architecture running, with a real-time operating system and put to practise all the research made in this area by the group.

The problems approached with this work, required a broad understanding of C, RTEMS operating system, and an in-depth knowledge on the processes of creating and configuring an RTEMS application. The research on the innerworkings of RTEMS is exposed in section 4.

2.1.1 Circular Buffer

With the plan of understanding the behaviour of RTEMS on an SMP architecture, the idea of developing a feature that would register the preemptions appeared, easing the verification of the new test-suites and help the team to understand the behaviour of the tested mechanisms.

This feature turn to be a circular buffer that saves the last one thousand preemptions, it is initiated when a RTEMS application is executed, being supported for any scheduling algorithm that supports Symmetric Multiprocessing. A new RTEMS directive was also developed to present the content when requested by the application.

For the development of this buffer, an extensive comprehension of RTEMS kernel was required.

2.1.2 RTEMS Test Suites

RTEMS, having his code publicly available allows for everybody with interest and capabilities to be able to study and learn about RTOS, RTEMS, and all sorts of techniques involved in designing and implementing such a specific system like this one.

The test suites that are also publicly available, found on the RTEMS GitHub page together with the rest of the source code, are a set of RTEMS applications, created usually by people that were either involved in the development of this RTOS or that are in some way related with this field of work and revealed interests in learning and developing an application that targeted a specific part/functionality of the operating system.

The development of RTEMS in SMP in an ongoing project and as mentioned before, two of the most important parts the RTEMS operating system (scheduling and synchronization) lacked the presence of SMP test suites that targeted its behaviour.

This tests not only attest the behaviours expected or documented but can also be used as example applications for future developments.

2.1.3 Real-Time Case Study

The “mine control system” is arguably a suitable use of a very well-studied case study, in an ARMV7-A multiprocessor platform. With the multiprocessor platforms being so fresh on the embedded world, its expected not to find many implementations thoroughly analysed. We look to address this with the mine control system implementation which is a case study that represents exquisitely well a typical real-time system environment.

This case study addresses the supervision of a water pump placed inside of a mine. The purpose of this pump is to keep the water level on the mine inside a certain threshold. Besides the water level, there are also certain values regarding the quality of the air that are being monitored and that interfere with the functioning of the pump (e.g. if the CO level is above a certain level, the pump cannot be turned on). The whole system is developed with the intention to correctly operate the water pump, respecting all the restrictions that one might suffer and meeting all the deadlines imposed.

2.2 State of the art

On the last few years, several real-time operating systems (RTOS) developers have been working on bringing SMP support to their software. However, taking into consideration the number of active RTOS, those that support Symmetric Multiprocessing (SMP) are in a clear numerical disadvantage.

In 2011 it was presented the multicore edition of eT-Kernel, an OS for: “next-generation embedded systems with multi-core", “Ideal for high performance embedded systems such as digital home appliances, automobile devices, and mobile devices “[10].

As of the beginning of 2018 there are a few open source operating systems that brought this support for the embedded systems industry, like the one focused here RTEMS, and others (I.e., Nuttx, Nucleus RTOS and eCOS). When it comes to non-open-source software, VxWorks and QnxNeutrin. These are two of the most used RTOS and the ones with longest history, so it seems natural these were also one of the first to receive SMP support. (A full list of all depleted RTOS, and those still in use today is available at [7]).

There are currently some RTOS in beta phase. These are all open-source with the main purpose and characteristics of each varying:

- Simba
- SImpleAVROS
- SOOS Project
- Mark3

This work was meant to be developed on an ARMv7 based board, the next focus of analysis here is the RTOS RTEMS, which was the chosen operating system to run over this architecture. It is important to notice what changes the adaptation to SMP brings to the RTOS.

2.2.1 RTEMS

Before standing for Real-Time Executive for Multiprocessor Systems [8], RTEMS meant Real-Time Executive for Missile Systems, and before, Real-Time Executive for Military Systems when it started its activity working for the US Army in 1988, and it was in the beginning, like all RTOS developed until now, meant to be deployed on Uniprocessor systems.

The constant demand by applications for processing power along with the physical limitations faced by the semiconductor industries, fermented the appearance of more than one CPU and that was transposed to the domains of embedded systems.

Embedded Brains GmbH, one of the companies that is actively involved on the development of RTEMS, in the years before 2017 started working on the extension of the OS to support SMP configurations and on May, the same year, the software architect Sebastian Huber presented it for the first time at DASIA (Data Systems in Aerospace). As is it found on the RTEMS documentation: *"The RTEMS interpretation of real-time on SMP is the support for clustered scheduling with priority-based schedulers and adequate locking protocols."* [9].

The drive to develop the RTEMS support for SMP was laid on top of fact that traditional software implemented and designed with a uniprocessor architecture in mind do not scale and the trend towards the adoption of multicore processor platforms is evident in embedded systems, and even in the broader cyber-physical systems domain. The support for Symmetric Multiprocessing (SMP) came to solidify RTEMS as a state-of-the-art RTOS with the possibility to be implemented in hardware systems with true parallel processing capabilities.

The increased software and hardware complexity and the presence of true parallelism (which does not occur in uniprocessor) leads to the application developer having to be even more careful about mutual exclusion and shared data access. Problems that are rarely or never found on uniprocessor now appear and must be dealt with.

2.2.2 (New) SMP capabilities brought to RTEMS

→ Partitioned/Clustered Scheduling

In clustered scheduling the set of processors that constitute the system is partitioned into non-empty disjoint subsets, called clusters. Clusters that only contain one processor are called partitions and each cluster is owned by one scheduler instance. Unlike Asymmetric multiprocessing, in SMP there is no physical barrier separating the different subsets. There is only a logical barrier, but sometimes, in specific situations, tasks can run on processor subsets that do not belong to it scheduler instance.

Clustered scheduling helps to control the worst-case latencies of a system and reduces the amount of shared data in the system. Also, it was implemented for RTEMS SMP to best use the cache topology of a system.

→ Scheduler Helping Protocol (helping hand mechanism)

The schedulers implemented (not all) provide a helping mechanism to support the necessary locking protocols.

RTEMS is implemented in a way which each task has its own scheduler instance. One scheduler instance can be running several scheduler nodes (CPUs), consequently, the scheduler instance the task is attributed to determines the CPUs this task can run on. This is a clever mechanism but in a multiprocessor platform this can create some problems. The way RTEMS implementation bypasses these technical hurdles is by having a helping protocol, which allow for tasks to gain access to CPUs of other scheduler instances.

For the scheduler helping protocol to be available the following operations must be implemented by an SMP scheduler:

1. ask a scheduler node for help,
2. reconsider the help request of a scheduler node,
3. withdraw a schedule node.

Even though this helping protocol is necessary due to the locking mechanisms, the deployment details involve changing the scheduler kernel code, and this is one of the main reasons why locking protocols are still complex to adapt in different OS. Furthermore, this is precisely the reason why having locking protocols implemented (not even in use) causes an overhead on the main scheduling procedures [10].

“All currently available SMP-aware schedulers use a framework which is customized via inline functions.” [11]. This is a way to allow an easier implementation of scheduler variants.

→ Profiling

The support for profiling of low-level synchronization was added to allow for the identification of bottlenecks in the system and it is a tool that can be accessed by a build-time configuration flag. Profiling reports are generated in XML for most test programs of the RTEMS test-suite and were implemented with an acceptable overhead.

The number of tests already developed give a good sample set for statistics. One can know for example the maximum interrupt latency, or the lock contention latency.

→ Fine grained locking

Fine grained locking allows for a much less resource contention in a system because each object has its own lock to protect the object state. With a giant lock, we can have our example thread performing a certain job that requires mutual exclusion have their execution time affected by other thread who is on the same critical zone our thread is trying to access and yet in the end represent no real concurrency. This creates a real bottleneck in a system and its a problem that is addressed by designing a system of fine-grained locks, since the more fine-grained the less likely one thread will request a lock held by other.

In RTEMS fine grained locking was first implemented for events, semaphores and message queues and it was proven that this implementation scales well with the count of active tasks even outperforming the old implementation.

→ Time keeping (redesigned)

A solid, high performance timestamp implementation is crucial for the overall system performance, and specially for safety-critical real-time systems who are so dependent of time constraints. The extension used to get timestamps was broken by design on SMP, so a whole new implementation was necessary.

The possible solutions were equated, and the FreeBSD time counters were selected to achieve the desired solution, as they presented excellent results and are a lock-free solution.

The aim of development of RTEMS SMP was to maintain a low-overhead operating system suitable for safety-critical activities. Currently, the implementation presents some limitations, i.e., lack of support for locking mechanisms in dynamic priority-based schedulers, (even though is not a problem of the RTEMS SMP development, per se) nevertheless it presents a solid low-level implementation when it comes to:

- Low-level synchronization,
- thread migration and processor assignment,

- SMP scheduler framework,
- partitioned/clustered scheduling,
- thread queues (building block for objects which may block a thread), and
- thread-local storage.

Making it ready for production systems.

2.3 Business Areas

In very few words, cyber-physical systems are the systems that connect the physical world with the information processing world of the computer. But if we're trying to be accurate, what a cyber-physical system really is, is the set of computing elements which are responsible for or are related to the process of supervising physical sensors and actuators with the goal of retrieving information from the environment they are in and respond with certain actions. The wide success of CPS in today's world can be attributed to the very own embedded device, or embedded system. In fact, these two terms are almost glued together since in reality, cyber-physical systems build upon the "older" technology of embedded systems.

One of the first uses of a modern embedded system with the properties that we are familiar with (small size, "high" processing capabilities, …) can be traced as far back as 1961. This year marked the launch of the famous Apollo program, carried out by NASA, and it was when Charles Stark Draper started working on the riskiest piece of equipment of the entire program, the Apollo Guidance Computer.

Embedded safety-critical systems are usually designed to be very small, to have real-time capabilities and are often incorporated within a larger computer system. The economic and societal potential of these systems is enormous. The market for this kind of devices was estimated to be already over 120 billion euros in 2013 and major investments are constantly being made worldwide to develop the technology.

CPS has been identified as a core enabling and disruptive technology by the German National Academy of Science and Engineering (acatech) and the impacts and possibilities these systems bring to our lives are still subject of thorough studies.

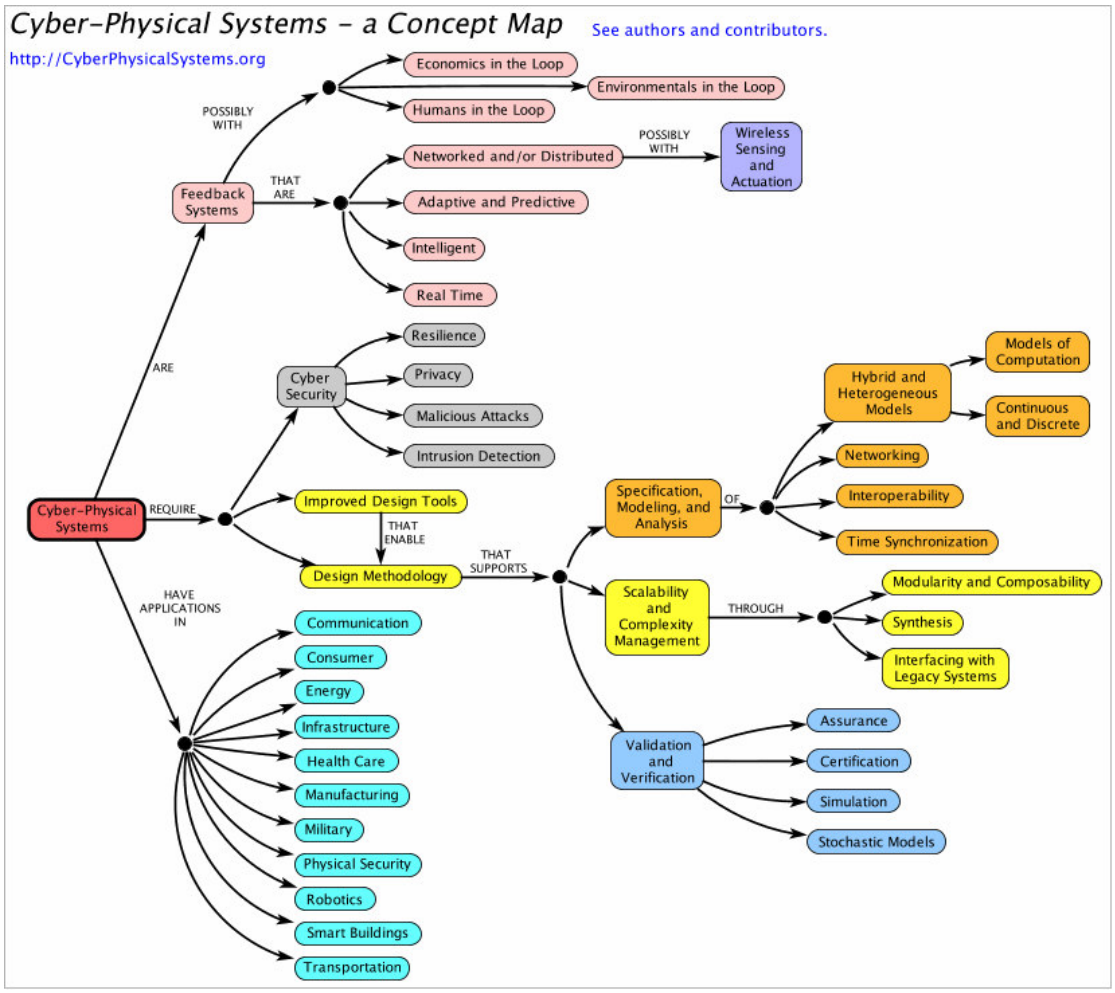


Figure 2 Cyber-Physical Systems Map
data source: https://ptolemy.berkeley.edu/projects/cps/

Perhaps the impact these systems have can be better understood by giving out some examples of the application of these systems in different fields of work:

2.3.1 Manufacturing

CPS are used to improve processes by sharing real-time information among all different levels of machinery and people working. Furthermore, CPS can improve these processes by self-monitoring and controlling the entire production processes and by allowing it to adjust production we achieve a higher degrev cxjgc,e of visibility and control on supply chains.

2.3.2 Healthcare

CPS are used in real-time and remote monitoring of the physical conditions or to help disabled and elderly patients. Besides, CPS are widely used in research in the

neuroscience field to better understand human functions. (for e.g. with the support of brain-machine interfaces and therapeutic robotics.)

2.3.3 Energy

CPS are used to monitor energy expenses. In a smart grid, for example, this technology is put into practise to allow for a more efficient use of the grid and to make it overall more transparent and sustainable. The smart cities are very closely related to this and extend even more the domain of action of these systems. With CPS you can enable energy monitoring and control systems usage, or you can determine the extent of damage that buildings suffer after unexpected events and help prevent structural failures.

2.3.4 Automotive

Individual vehicles can communicate with each other, sharing real-time information about traffic, location, or other issues and have the main reason behind it to prevent accidents or congestion, improve safety, and ultimately save money and time. Nowadays, the automotive industry actually consumes more embedded devices than any other industry. This is because modern cars dispose of a distributed system of devices that are used internally to manage all kinds of elements that integrate the vehicle.

2.3.5 Agriculture

CPS can be used to gather information about different aspects like the climate, the ground, and such, allowing for an accurate application of agricultural techniques. A CPS can also be constantly monitoring different resources, such as watering, humidity, plant health and others, through sensors and, thus, keep the ideal environmental values without the help a supervisor.

2.3.6 Computer Networks

CPS can boost cyber environments to better understand systems and users' behaviours, which can help improve performances and resource management. Moreover, popular social networks and e-commerce websites store users' navigation information and users' web content, analyses that information, and then tries to predict interests and make recommendation for friends, posts, links, pages, events, or products.

There are many more subtle ways CPS are present in our lives and businesses are already developing or contributing towards CPS even sometimes without realising it.

There are many parts that constitute a cyber-physical system, this includes the software needed to run the system(s), the set of sensor and actuators, the communications technology used. And a CPS often includes components from many different manufacturers or service providers.

CPS presents a collection of challenges not always found in a classical information or embedded system. Mastering the engineering of complex and trustworthy cyber-physical systems is important for allowing our industries to implement CPS-based business models which could bring unprecedent benefits for companies and consumers. Current CPS, however, are still very expensive to develop and maintain and sometimes with unknown repercussions.

3. Working Environment

This chapter relates the timeline and how the project developers worked, explaining the work methodologies and the technologies used. For a better support the planning and a Gantt diagram will also be included in this chapter.

3.1 Work Methods

Since the beginning it was considered that the best work method, for his own nature, should be a linear sequential design approach, specifically waterfall model, being less iterative and flexible.

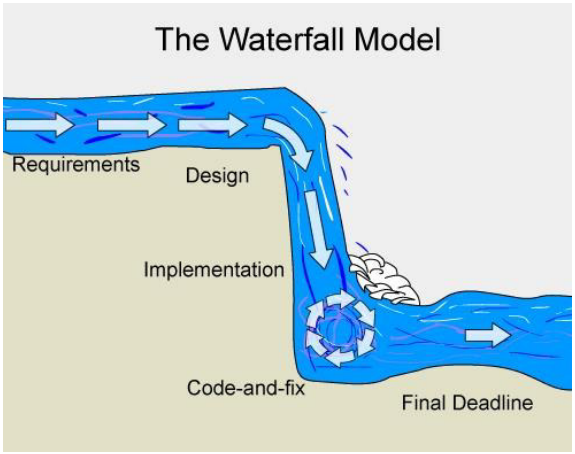


Figure 3 Waterfall model representation
data source: https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology

The project was divided in three development phases, in which the waterfall method was used on the last two. The first one consisted in a deep initial study of the operating system and in the integration of the same on a hypervisor, QEMU, that allowed the team to have a total hardware abstraction, the second phase to develop the samples and the buffer, the third development phase was dedicated to the real-time case study. The team was unable to make use of, maybe, more modern or flexible approaches as these put an emphasis on the use of model artefacts (visual representations) and full cycles of testing phases, that do not fit well with the working of this project.

For the first one, as the waterfall method purposes, the timeline was developed in the following steps:

1. Analysis: This phase consisted on an extensive studying of the operating system.
2. Design: During this phase the domain analysis was developed, such as the system architecture of the circular buffer.

3. Implementation: In this phase all requirements both for the buffer and for the samples were implemented.
4. Verification: End of the first development state, the studying and comprehension of RTEMS, in this phase the team verified if both the design and implementation were correct and functional, as well as notify the anomalies of the operating system.

As this was a research phase and full of unknown technologies and subjects, the analysis and design steps had a much more meaningful part than in typical software development.

And for the case study:

1. Requirements Analysis: Comprehension of the case study and all is requirements.
2. Design: In this phase the physical and logical architecture design were developed.
3. Implementation: Implementation of the requirements.
4. Testing: End of the project, testing and conclusion of the case study.

For this project it was used bitbucket to manage the source code and its design, concerning the tasks/issues, the team used Trello as it can be seen in appendix 6.2, these work methods allowed to see what each member of the team was doing, had done and what was going to do.

3.2 Work Planning

The project was divided in 9 phases: Studying and the Comprehension of the behaviour of RTEMS, integration on an emulator, kernel and samples analysis/design, kernel development, samples development, case study analysis and design, case study development and Documentation. It was decided to not include the meetings, since they occurred almost on a daily basis or when doubts appeared, these meetings occurred with the senior engineers of CISTER.

The most important phases were the comprehension of RTEMS and the kernel and samples analysis, the first one allowed to almost fully understand the behaviour of the operating system and in a following phase, the kernel and samples analysis allowed us to study the code of RTEMS, which gave an overview for a later development of the circular buffer on the Kernel and the samples. This allowed the team to meet the established goals for the comprehension of RTEMS.

The shortest phase was the integration phase, that consisted on placing the Operating System working on an emulator and to enable certain configurations to simulate an

embedded environment, for that, a couple of technologies were used, the QEMU and RTEMS Source Builder.

And for last, the case study allowed the team to evaluate their knowledge and to gain a practical overview. In both development phases the team implemented all previous requirements.

The documentation phase contemplates the writing of the report for PESTI and a paper for CISTER.

TASK	DURATION	START	FINISH
RTEMS STUDY	4 Weeks	26 February	26 March
INTEGRATION	2 Weeks	26 March	9 April
KERNEL/SAMPLES ANALYSIS/DESIGN	4 Weeks	9 April	7 May
KERNEL DEVELOPMENT	4 Weeks	30 April	21 May
SAMPLES DEVELOPMENT	4 Weeks	21 May	18 June
CASE STUDY ANALYSIS	3 Weeks	18 June	9July
CASE STUDY DESIGN	2 Weeks	2 July	16 July
CASE STUDY DEVELOPMENT	5 Weeks	16 July	20 August
DOCUMENTATION	26 Weeks	19 March	14 September

Table 1 Project Planning

Table 1 reveals the project timeline for the 9 phases of the work, Appendix 11.1 shows a complete overview of the Gantt Diagram.

3.3 Technologies

A multitude of technologies were used to develop the current work, without those it wouldn't be possible to satisfy all requirements, table 2 summarizes those employed technologies.

TECHNOLOGY	USE
C	RTEMS Kernel and Samples
QEMU	Emulator
SOURCETRAIL	Indexation of the code
GNOME COMMANDER	"Two-pane" graphical file manager
RSB	Build RTEMS compiler and OS
RTEMS	Operating System

Table 2 Employed technologies

The following subsections will be a small briefing on what each of these technologies consists and how they served a use for the project.

3.3.1 C (Programming Language)

C [12] is a general-purpose, imperative computer programming language ideal for developing firmware, operating systems, language compilers, assemblers, network drivers and portable applications, supporting structured programming, lexical variable scope and recursion. It was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs and used to re/implement the Unix operating system.

C has several important features such as:

- Fixed number of keywords, including a set of control primitives.
- Multiple logical and mathematical operators, including bit manipulators.
- Multiple assignments may be applied in a single statement.
- Function return values are not always required and may be ignored if unneeded.
- All data has type but may be implicitly converted.
- Basic form of modularity. And control of function and object visibility to other file via extern and static attributes.

In this project C was used to develop both the RTEMS Kernel and Samples, taking advantage of his features and properties.

3.3.2 QEMU

QEMU (Quick Emulator) [13] is a hosted hypervisor that performs hardware virtualization, it simulates CPUs through dynamic binary translation and provides a set of devices models, enabling it to run a variety of unmodified guest operating systems, QEMU has two operating modes, the full system emulation and the user mode.

In the full system emulation, QEMU emulates a full system (such as a PC) including one or several processors and various peripherals, this mode can be used to launch different operating system without rebooting the PC or to debug system code, it also has other features such as: a full software MMU for maximum portability, an in-kernel accelerator and a symmetric multiprocessing support.

And finally, the user mode emulation allows to launch processes compiled for one CPU on another CPU or to ease the cross-compilation and cross-debugging, this mode also allows features such as: a generic Linux system call converter and a signal handling by remapping host signals to target signals.

In this work QEMU was used to virtualize the hardware allowing to boot the operating systems RTEMS, the following figure represents the QEMU architecture used in this project, that is, a guest operating system working on a host operating system.

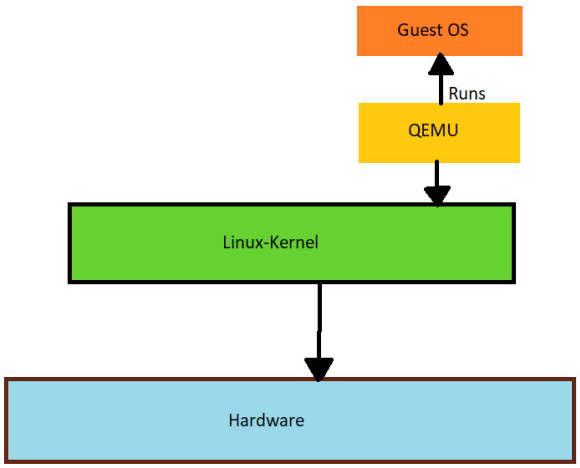


Figure 4 QEMU architecture with guest OS

3.3.3 SOURCETRAIL

Sourcetrail [14] is an interactive source explorer that simplifies navigation in existing source code, supporting several languages such as C/C++ and Java, it first indexes the code and gathers data about its structures and then provides a simple interface consisting of three interactive views, each of one plays a key role in getting information.

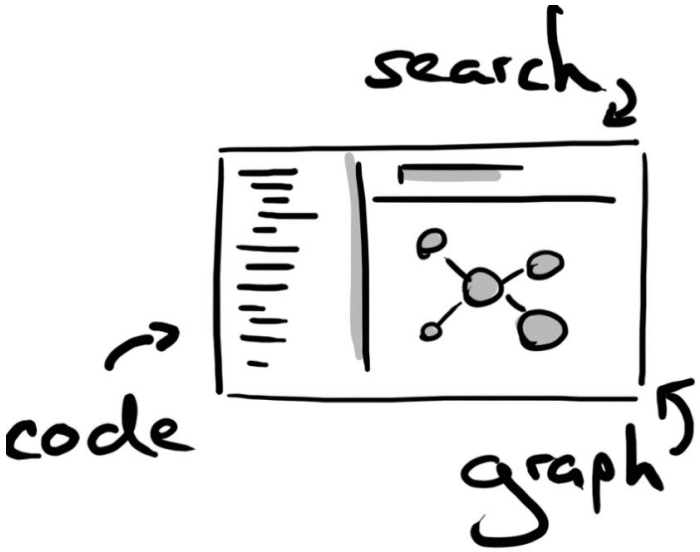


Figure 5 the 3 interactive views of Sourcetrail
data source: https://www.sourcetrail.com/

- **Search**: allows to quickly find and select indexed symbols in the source code, instantly providing an overview of all matching results.
- **Graph**: displays the structure of the source code, focusing in the selected symbol and directly showing all incoming and outgoing dependencies to other symbols.

- **Code**: displays all the source location of the selected symbol in a list of code snippets.

This technology was used to clarify the source code and the dependencies of RTEMS, which allowed to a faster and better comprehension of the operating system.

3.3.4 GNOME Commander

GNOME Commander [15] is a “two panel” graphical file manager for Linux desktop, it aims to fulfil the demands of more advanced users who like to focus on file management and has the following features:

- MIME TYPES.
- Network support through FTP, SFTP.
- User defined context menu.
- Plugin support.
- Python scripting.

GNOME Commander was used to simplify the file management of this project, allowing an easier navigation within the directories and files of RTEMS.

3.3.5 RSB (RTEMS SOURCE BUILDER)

RTEMS Source Builder [16] is a tool to build packages from source, it is used by the RTEMS project to build its compilers and OS, it helps consolidate the details needed to build a package from source in a controlled and verifiable way, RSB can also build bare metal development environments.

RSB has been tested on several OS, such as:

- ArchLinux
- CentOS
- Fedora
- Raspbian
- Ubuntu
- Linux Mint
- openSUSE
- FreeBSD
- NetBSD

- MacOS
- Windows.

The RTEMS Source Builder has two type of configuration data, the first is the build set. A build set describes a collection of packages that define a set of tools you would use when developing software for RTEMS, for example the basic GNU tool set is binutils, gcc and gdb. These are a typical base set of tools needed for an embedded cross-development type project.

The second type of configuration data is the configuration file and they define how a package is built, configuration files are scripts loosely based on the RPM spec file format and they detail the steps needed to build a package.

As is it explain before, the RSB was used in this project to build RTEMS compiler and OS.

4. RTEMS

As mentioned on chapter 2.2, RTEMS Real-Time Executive for Multiprocessor Systems [8] is a Real-Time Operating System that supports open standard applications programming interfaces (API) such as POSIX, ADA, native and ITRON. It is used in space, flight, medical, networking and many more embedded devices using architecture including ARM, PowerPC, Intel, Blackfin, MIPS, Microblaze, SPARC and many others.

In POSIX terminology, RTEMS implements a single process, multithreaded environment. With the existence of only one address space, all flows of control (threads) share the same address, turning it into a closed real-time system, where only one application is started when the RTEMS is switched on. RTEMS closely corresponds to POSIX Profile 52 which is “single process, threads, filesystem”.

		Process	
		Single	Multi
File System	No	51	53
	Yes	52	54

Figure 6 POSIX Profiles
data soure: http://www.opengroup.org/testing/testsuites/POSIXProfiles.htm

RTEMS can be considered as a set of layered components that provides services to a real-time application. The interface presented to the application is formed by joining directives into logical sets labelled resource managers. The following figure shows the managers organization.

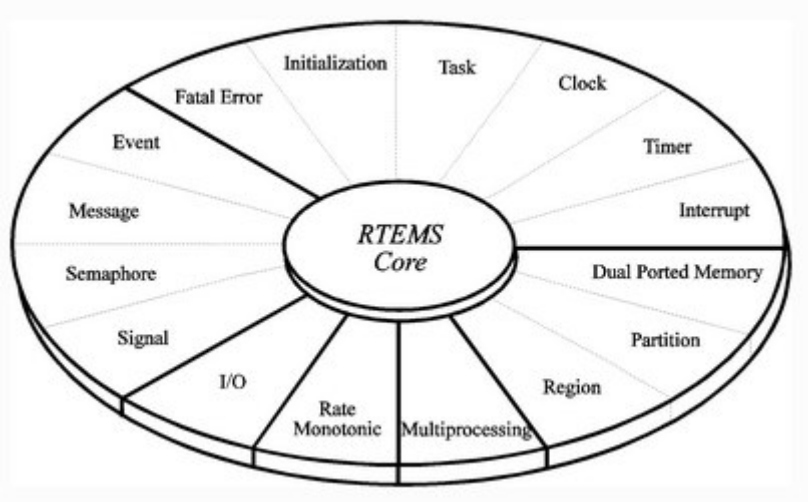


Figure 7 RTEMS Organization
data source: https://docs.rtems.org/branches/master/c-user

RTEMS Core depends on a small set of processor dependent routines, being part of the executive core functions such as scheduling, dispatching and object management, that are used by several managers.

In the following sections, the layers, principal managers and functionalities of RTEMS will be further explained.

4.1 RTEMS Layers

RTEMS is characterized by three layers: hardware support, kernel and APIs, the user can develop his application by using available APIs, as it can be seen in figure 8.

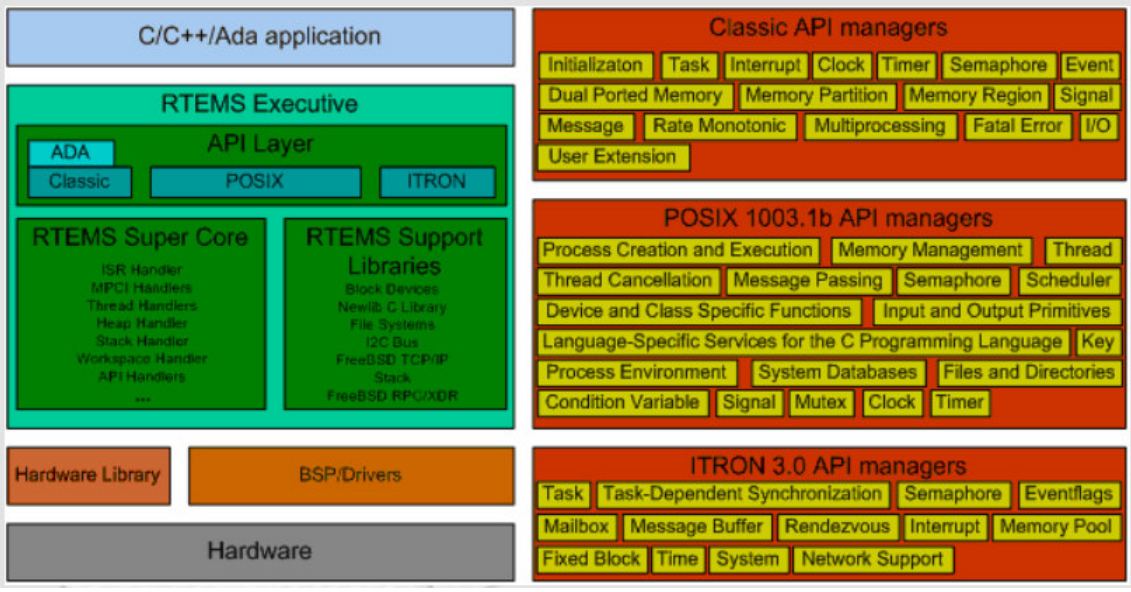


Figure 8 RTEMS layers
data source: https://docs.rtems.org/branches/master/c-user

The hardware support layer encompasses the processor and board dependent files as well as a common hardware library. One aspect that can be noticed is that both the kernel and the API layers are part of the so called RTEMS executive. The notion of executive expresses the capability to run applications, implying the use of an API set for application development, however, from a conceptual level the kernel itself and the APIs are two distinct ideas.

The kernel layer is the heart of RTEMS and encompasses the super core , the super API and several portable support libraries. The super core is organized into handlers and provides a common infrastructure and a high degree of interoperability between APIs. The super API contains the code for services that are beyond any standardization, such as API initialization and extensions support.

The API layer makes the bridge between the kernel and the application. The Classic, POSIX and ITRON APIs are implemented in terms of super core services. Each API is organized into managers (the right side of the image illustrates that). The Ada API is a direct mapping of the Classic interface.

4.2 Floating Point

RTEMS provides software and hardware floating-point support, the presence or absence of the RTEMS floating-point unit (FPU) attribute in the creation of a task determines whether it is floating-point enabled or not. When creating a task with the FPU attribute flag results in additional memory being allocated for the task control block (TCB) to store the state of the numeric coprocessor during task switches.

Saving and restoring the context of a task with FPU takes longer than a task that does not have FPU defined, mainly because of the relatively large amount of time required for the numeric coprocessor to save and restore its computational state.

If the supported processor type does not have hardware floating capabilities or a standard numeric coprocessor, a FPU emulation software library must be utilized for floating-point operations, or else all the task will be defined as no floating-point.

4.3 Task Manager

This manager provides a comprehensive set of directives to create, delete and administer tasks. But for the real understanding of this manager several definitions must be explained, for start, what is a RTEMS task?

A task, in RTEMS perspective, is the smallest thread of execution which can compete on its own for system resources, moreover, each task is established by the existence of a task control block (TCB). TCB is defined as a data structure which contains all the proper information to the execution of the task, it is allocated upon the creation of the task and released when the task is deleted.

The directives offered by this manager, allows the application to create tasks, by allocating the TCB, stack and floating-point context area. All created tasks are initially placed in the dormant state. The start operation places a dormant task into the ready state, being initialized the task's stack upon the task's initial execution mode and start address, meanwhile the restart directive restarts a task at its initial starting address with its original priority and execution mode, but with a possibly different argument. When suspending a task, it is passed to a blocked state until the resume directive is called, placing it into the ready queue.

To remove all references to the task, RTEMS provides a directive, this operation frees the task's control block, removing it from resource wait queues, and deallocates its stack as well as the optional floating-point context.

All the states transition associated with the directives from this manager are shown in figure 9.

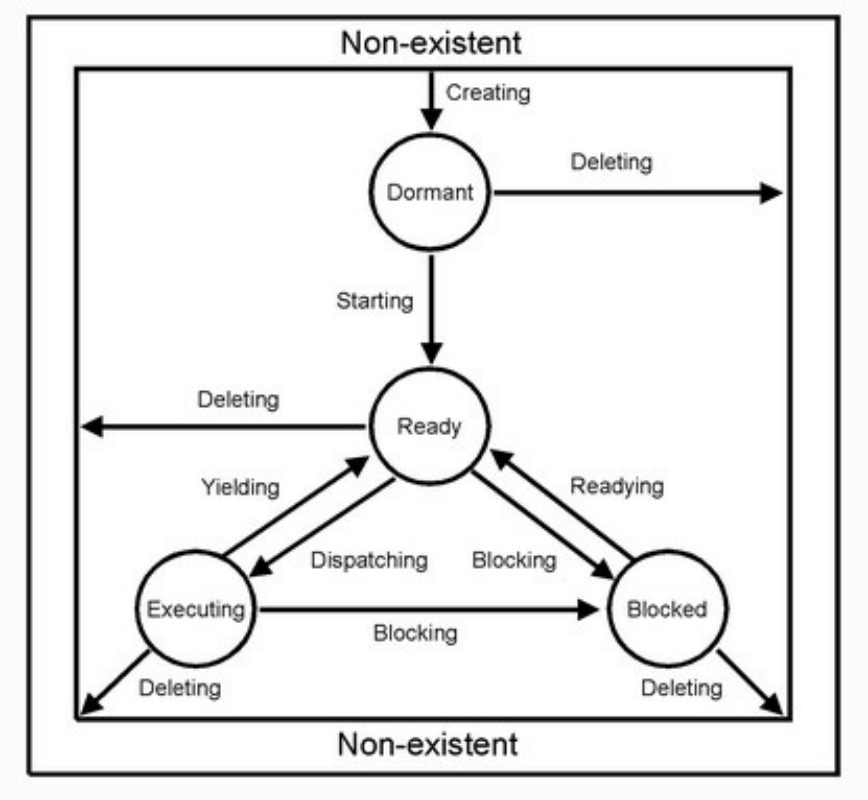


Figure 9 RTEMS Tasks States
data source: https://docs.rtems.org/branches/master/c-user

For the creation of tasks with periodic fashion, I.e., jobs of a task appear in the system with a regular interval, the Rate Monotonic Manager is used, it helps creating tasks with periodicity by defining a period with an operation from this manager. It also gathers information about the execution of those periods and can provide statistics to the user which can be used to analyse and tune the application. The services provided by this manager may be used by any application which requires periodic tasks.

4.4 Timer Manager

A timer is an RTEMS object which allows the application to schedule operations to occur at specific times in the future. For the use of timers, it is required the use of a clock tick. User supplied timer service routines are invoked by either a clock tick directive or a special Timer Server task when the timer fires. Timer service routines may perform any operations or directives which normally would be performed by the application code which invoked a clock tick directive.

4.5 Interrupt/Signal Manager

RTEMS, as any other real-time operating system, provides mechanisms for response to generated interrupts. For that, it offers the Interrupt Service Routine (ISR) and the Asynchronous Signal Routine (ASR), both formats are extremely similar.

ISR represents hardware interruptions, a software process is invoked by a hardware device, it allows the application to connect a function to a hardware interrupt vector, when an interrupt occurs, the processor will automatically vector to RTEMS, saving and restoring all registers which are not preserved by the C calling convection, giving the control to the ISR, meanwhile, ASR, represents software interrupts. In other common words, asynchronous signal routine is to a task what an ISR is to an application's set of tasks, when a signal is sent to a task, the task's execution path will be changed by the ASR.

There are several differences between those software and hardware interruptus, such as:

- While ASR are scheduled by RTEMS and can invoke any directive, ISR are scheduled by the processor hardware and can invoke only a set of directives.
- ISR are immediately handled, while the ASR are only handled when the receiver task enters the processor.

4.6 RTEMS Scheduling

As mentioned before, for real-time systems, the scheduling dictates the ability to provide immediate response to external events. Scheduling functions do not have a specific manager, belonging to the RTEMS core.

RTEMS provides a plugin framework which allows it to support multiple scheduling algorithms, both for uniprocessor and for SMP, the application can select at link-time which of these to use, being more appropriated to the specific use case.

As the project focused on SMP, only SMP scheduling algorithms will be further explained. All SMP schedulers are priority based, the processors managed by a scheduler instance are allocated to the highest priority tasks allowed to run, the SMP scheduler are the followings:

- EDF, Earliest Deadline First.
- Deterministic Priority
- Simple Priority
- Arbitrary Processor Affinity Priority.

4.6.1 Earliest Deadline First (EDF) SMP

Earliest deadline First is a dynamic priority scheduling algorithm used in real-time systems, where the priority of the threads can change during its execution, it is also the default scheduler in SMP configuration if more than one processor is configured.

The concept of a deadline shorter than the period, or explicit deadline, do not exist within RTEMS, only implicit deadline are admissible, so, when creating a periodic task, the given periodicity will correspond to its deadline, maximum time where a response must be guaranteed.

With EDF for RTEMS there is an attribution of two kind of priorities to tasks, to the background tasks, that is, tasks that do not have any periodicity, the maximum priority is given as $\min(INTMAX, 2^{62} - 1)$ and its attributed by the application being a fixed priority. Meanwhile, tasks with an active period have higher priority than the background tasks, being defined a higher priority to the tasks with closest deadlines, that is, the priority is inversely proportional to its deadline.

EDF RTEMS supports task processors affinities of one-to-one and one-to-all, in common words, it associates the tasks with the processors, allowing a task to run uniquely on one processor or in all.

4.6.1 Simple Priority SMP

Simple Priority SMP is a fixed-priority preemptive scheduler which uses a sorted chain for the ready tasks, placing in the processor the task with the highest priority. This scheduler and all the other fixed priority schedulers are more controllable and predictable than the EDF scheduler.

By convention, the maximum priority level is 255, but in RTEMS the implementation limit is $2^{63} - 1$.

4.6.2 Deterministic Priority SMP

The Deterministic Priority SMP is a fixed priority preemptive scheduler commonly used in real-time systems, it executes the highest priority task of all the tasks that are in the ready state.

This scheduler is extremely similar to the Simple Priority SMP Scheduler, but diverge with the chain that is used, while the Simple uses a unique chain, the Deterministic uses table of chains, with one chain per priority level for the ready tasks, then applying FIFO in each chain.

The maximum priority level is configurable, nevertheless, by default the maximum priority level is 255.

4.6.3 Arbitrary Processor Affinity Priority SMP

Arbitrary Processor Affinity Priority SMP is just like the Deterministic Priority SMP, a fixed-priority scheduler which uses a table of chains with one chain per priority level, it also as the same maximum priority level and configurability as the Deterministic. The main difference between both, is that, this scheduler supports arbitrary task processor affinities, allowing a task to execute only on certain processors, depending on the configuration.

The affinity is changed through the variable *cpu_set_t* that represents the affinity set, each bit corresponds to a processor, a set bit means the task can execute on this processor and a cleared bit means the opposite.

4.7 RTEMS SMP Synchronization Mechanisms

The capacity for synchronization and communication between the different running tasks in a system is a fundamental requirement to be able to control the execution of said tasks, and thereby, the system.

In uniprocessor, RTEMS offers different mechanisms that have been thoroughly analyzed and designed to allow an efficient way for thread synchronization and communication. These are:

- Semaphore Manager
- Message Manager
- Event Manager
- Signal Manager

The semaphore manager supports mutual exclusion capabilities, involving the synchronization of access to one or more shared user resources. The message manager supports both communication (tasks can send messages to each other) and synchronization (tasks can be put on hold, waiting for a message to arrive). The event manager primarily provides a high-performance communication (through event sending) and synchronization mechanism (through putting a task waiting for a certain event to arrive). The signal manager supports only asynchronous communication and is typically used for exception handling. The low-level synchronization primitives used on RTEMS were implemented using C11 atomic operations.

The SMP lock, a ticket-lock, implemented in RTEMS uses FIFO ordering, since this is meant for systems in which high predictability is a necessary quality, even more than high throughput. The RTEMS API is also capable of supporting MCS locks, with the purpose of allowing the OS to support more than 32 cores (in the future).

The following options are provided by:

- Events,
- message queues,
- semaphores
    - mutexes using OMIP,
    - mutexes using MrsP,
    - binary and counting semaphores.

The main differences in terms of synchronization from the uniprocessor platforms, are centered around the use of semaphores. As memory is shared among the different CPUs

of a system it is imperative to provide safe ways to do it (meaning: no corrupted data, provide mutual exclusion primitives). However, having the same semaphore being accessed from different CPUs leads to a noticeable worst-case execution time increment when performing the schedulability analysis.

Not only that, when working in multitasking, preemptive environments and shared resources, there is the possibility of a problem called priority inversion. In multiprocessor platforms this problem is exacerbated by the introduction of true task parallelism.

So, the operating system must implement a mechanism to deal with this. (what is known as a locking protocol.)

4.7.1 Priority Inversion

Most real-time operating systems employ priority-based preemptive scheduling algorithms as this is a good way to encode in a system the priority of tasks (we control the order in which tasks must be ran). These schedulers attribute a priority level (number) to each task, and the higher priority tasks expect to run as soon as they are made ready to run.

Priority Inversion is the name given to a famous problem, that happens when the execution of a high priority task is interrupted by the execution of a lower priority task. This is how the situation can happen in a multiprocessor environment:

Tasks need to share resources to communicate and process data, and often times the shared resource must have a mechanism to grant mutual exclusion, (usually a semaphore) since the value of one variable cannot be changed at the same time, by two different tasks. There is a possibility that a low priority task – Task1, running on one processor, is made ready and accesses the shared resource used by a high priority task – Task2, running on another processor, and so Task2 must now wait for the low priority task to finish running (at least, on the critical zone). The time spent in the critical zone does not usually extend much in time, so it does not seem a very serious for Task1 to interfere with Task2 (even though it can be).

But the real problem happens when Task1 is preempted by a medium-priority task – Task3, at this moment, a priority inversion problem is said to occur. This would leave the higher-priority task in a pending state, waiting for other tasks with lower priority to run.

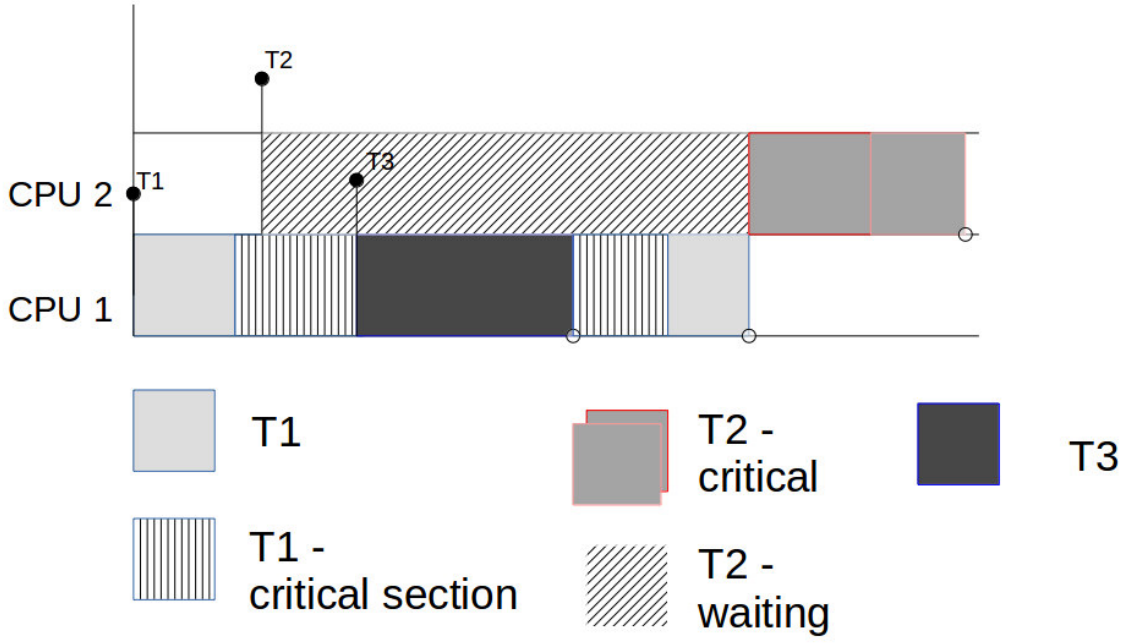


Figure 10 Priority inversion - example

Figure 10 illustrates the example of a typical priority inversion problem. We can see that when Task2 is ready to run, Task1 is executing the critical section. Task3 then enters the processor to run and since it does not make use of the resources shared, it will simply preempt the low priority task and run, leaving Task1, and consequently Task2 – high priority- on hold. Which is an unanticipated behavior when designing the system.

4.7.2 RTEMS Locking protocols

If priority inversion problems were to happen in real-time systems, as they sometimes do (a very notorious example was the 1997 Mars Pathfinder mission), the consequences would be fatal. Having no mechanism to go around this problem, it would not be possible to grant that higher priority tasks would run when they should, keeping up to their deadlines. In critical real-time systems this might mean a total system failure.

Research on this problem has been made and the shared resource protocols (locking protocols) are well studied and documented for uniprocessor solutions. For multiprocessor platforms there have been a large variety of protocols proposed, but most present serious setbacks. Besides the inherently high complexity of their implementation, either the protocol would impose restrictions to the synchronization primitive (i.e., Multiprocessor Priority Ceiling Protocol (MPCP) no possibility for nested resources) and/or it would bring significant run-time overhead, to systems that are usually very latency-sensitive [17]. It is important to mention that currently, no locking protocol was proposed that deals with dynamic priorities. The multiprocessor

semaphore locking protocols are usually designed having in mind a specific scheduling policie.

One first approach taken was to stop tasks, that are running on critical sections, to be preempted. However, this is not anywhere near an optimal solution, as it would lead to high priority tasks to be blocked more times than necessary (locket mechanism with big granularity).

During the RTEMS-SMP development only two locking protocols were found to have optimal characteristics for real-time multiprocessor environments [18]. They were developed for fixed-priority based schedulers and they are the Multiprocessor Resource Sharing Protocol (MrsP) and O(m) Independece-preserving protocol (OMIP). These protocols were published in 2013, the concept being generalizing the principles adopted by the Priority Ceiling Protocol (PCP) and Priority Inheritance Protocol (PIP). This report focuses in more detail on the earlier mentioned.

4.7.2.1 Priority Ceiling Protocol (PCP)

Priority Ceiling Protocol, also known as, Immediate Ceiling Priority Protocol (ICPP) attributes a ceiling priority to a mutual exclusion semaphore at creation time. Subsequent tasks that will acquire the mutex will have their priorities raised to the ceiling priority of the mutex. The ceiling priority must not be attributed randomly. This value should be set to the highest priority of the task that will ever attempt to obtain the mutex. Even though this protocol is beneficial for schedulability analysis, the need to identify the highest priority task that will ever attempt to obtain the mutex might prove very difficult in a more complex system.

4.7.2.2 Priority Inheritance Protocol (PIP)

With the priority Inheritance Protocol, the task that holds the mutex inherits the priority of the higher priority task that is trying to obtain the mutex. This inheritance is transitive, which means, if it happens to be a case where exists nested access between 3 tasks: Task1 is waiting for Task2, and Task2 is waiting for Task3 and Task1 has the highest priority, then Task3 will inherit the priority of Task1. This protocol does not however prevent the appearance of deadlocks completely.

4.7.2.3 Multiprocessor resource sharing protocol (MrsP)

The MrsP (stylised with lower-case not to be confused with the other, older protocol – MRSP) was published on a paper from 2013. Even though the aim of the people that worked on this protocol was to create "a general-purpose protocol that is applicable to (...) globally scheduled systems using fixed priorities, EDF or any other designation" [17] when published in July, their considerations were kept to fully partitioned systems using fixed-priorities.

At the time of writing, there is no protocol as developed as the MrsP or OMIP for the EDF scheduler, hence the lack of protocol implementation for this default scheduler in RTEMS. This protocol takes some characteristics of MRSP and builds upon it.

MrsP was developed to not block tasks that wait for a resource, instead, they perform a busy wait, as the alternative (suspension-based waiting) could mean longer waiting queues for tasks. Even though it is clear in the paper written that MrsP is supposed to have different ceiling priorities for each processor in the system with tasks that access the resource, currently RTEMS only allows for users to define a single priority ceiling (that is the same across all processors).

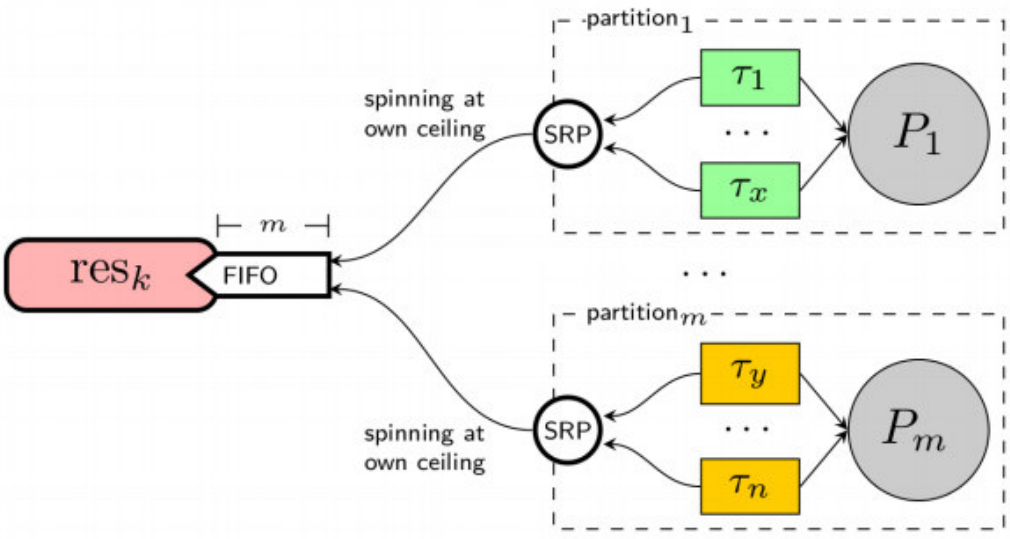


Figure 11 MrsP schema [18]

Image 11, taken from the RTEMS SMP final report depict how the connection and wait mode of the tasks waiting for the primitive, from different partitions is made.

The research made lead to the following protocols characteristics: When tasks ask to access a resource, they will have their priorities raised to the ceiling priority of the synchronization primitive they were trying to obtain. If the resource is in use by another task, then tasks wait in a FIFO queue, spinning at local ceiling priority. Note the detail.

At first glance it may seem to go against the good practices learnt to have tasks busy-waiting for a resource, but, here lies the distinctive feature of MrsP - a helping mechanism that allows for tasks to service the resource they are trying to use. This means, while tasks are busy-waiting for, let's say, Task1 to release the mutex, they won't leave the processor, and if it happens for Task1 to be preempted by a higher priority task, these "wasted" cycles of other mutex contenders can be used to run Task1 on its critical section and release the synchronization primitive. When returning to the processor, T1 will have their execution resumed after the mutex release part.

4.7.2.4 O(m) Independence Preserving protocol (OMIP)

OMIP is an Independence preserving protocol. This means tasks that are contending to access the mutex primitive won't be delayed due to unrelated critical sections accesses. This is a very desirable feature for real-time systems. And is not granted for example on the MrsP protocol. This protocol is aimed at clustered job-level fixed priority schedulers and, again, unlike MrsP which as a spin base waiting discipline, this has suspension-based locking.

This protocol aims at fulfilling the most desirable characteristics a real-time semaphore protocol must have, (according to the algorithm developers):

- The delays brought to tasks regarding the locking mechanism to be kept as low as possible
- keep high priority tasks unaffected by unrelated critical sections of lower priority ones.

The waiting queues that OMIP implements are more complex than the ones MrsP uses, but on clustered scheduling the behavior of both protocols is quite similar.

It is referenced on the RTEMS SMP final report that, even though OMIP offers promising characteristics, it is a recent, complex protocol and most usually known schedulability tests do not support its analysis. Even though MrsP has a much simpler design, the helping mechanism implemented has in practice some limitations [19].

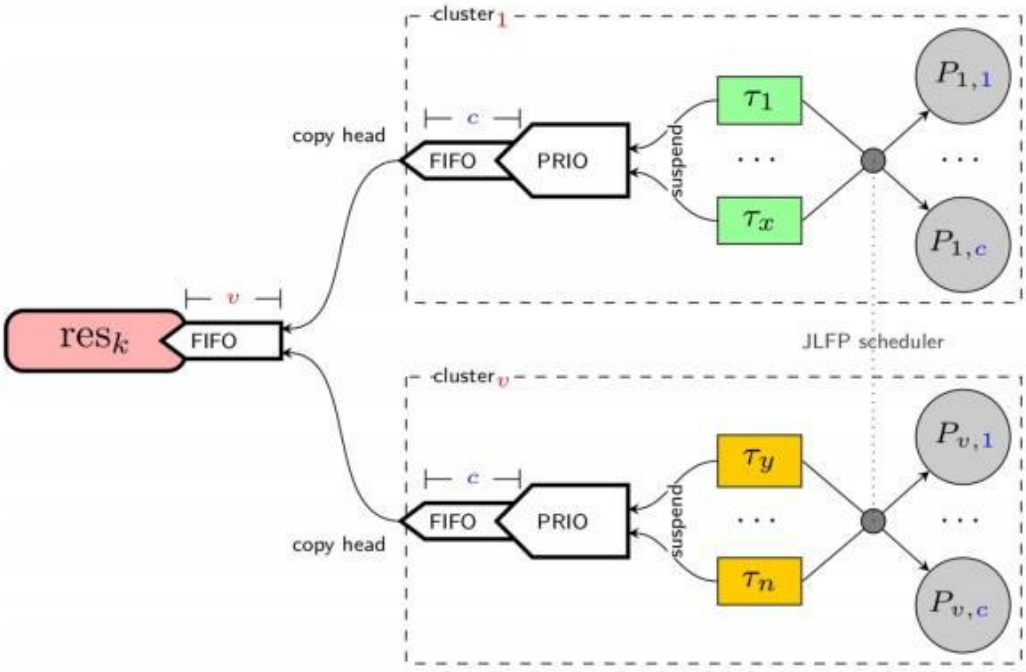


Figure 12 OMiP schema [18]

On figure 12 we can better have an idea on how OMIP operates. Note that, when tasks are waiting for the resource, they are suspended and put on a priority queue.

The defining characteristic of OMIP, is based on other protocol, the multiprocessor bandwidth inheritance protocol (MBWI) and is that lock-holding tasks may migrate freely among other processors where other tasks are waiting for the lock. The argument made on the paper [20] is that priority boosting is not very well suited for latency-sensitive systems, and they complement that with a small example. And so, another mechanism is proposed. This was the way found to get around that problem.

Following this approach, it was proven, when developing OMIP that it is impossible for a protocol to preserve the independence of the tasks, while still maintaining the priority inversion lock times acceptable and avoiding inter-cluster migrations.

This means the inheritance protocol is extended to clustered scheduling by introducing the concept of migratory priority inheritance. This works by keeping preempted, lock-holding tasks moving among clusters and leaving them in a cluster where a task is waiting to access the resource. The lock-holding task will then inherit the priority of the waiting-task and run on the foreign cluster, releasing the lock. Hence the utility of the FIFO queue present on the left side of the image.

4 RTEMS Build, Configuration and Installation

The initial project proposition contemplated, as mentioned before, the use of the RTEMS operating system dressing a multiprocessor platform. For this end, the RealView PBXA-9 board, based on a ARMv7-A architecture, was chosen as this is a fast platform choice for multiprocessor software development.

The lack of a physical board where we could implement the solutions never represented a deterrent for this project, as the number of emulation tools nowadays grows bigger as time passes. By accepting the suggestions made by our guiding teachers and after some research the group chose the open-source QEMU as the emulation platform for this work. This is a very famous emulation software/project that is even used by other emulation apps like VirtualBox. Furthermore, it offers some interesting features, making it a fast, robust application and offering support to a large number or architectures.

QEMU was used over 64 bits, Ubuntu 14.04 LTS operating system, this means the host machine is Linux. The targeted architecture, as stated before, is the ARM-V7 developed by the company Arm Holdings and licensed to a vast number of companies.

In order to run RTEMS we need to check out the source-code and compile it. To do this we make use of a tool already touched upon earlier, called RTEMS Source Builder, that downloads, builds and installs the compiler. Before that though, we needed to make sure that the source builder dependencies were all present and that every tool that we will be using later is installed. All configurations and installations were made using a Linux terminal.

To make way for an easier understanding of the code that will follow and that starts to be more technical, the directory tree of our project is now presented:

Project Directory Tree

$HOME/dev	The base directory used for software development
$HOME/dev/rtems	This project will be developed on the rtems folder
$HOME/dev/rtems/src	The rtems source code will be cloned here
$HOME/dev/rtems/compiler/5	Where the source builder source code will be checked out.

$HOME/dev/rtems/realview_pbx_a9_smp	Where the configuration for this specific board will be

Table 3 Project directories

5.1 Environment Setup

We start by installing QEMU, which, being available in the default repositories of Ubuntu, can be installed with:

```
$ sudo apt-get install qemu.
```

After that, it is necessary to make sure the system has a C/C++ compiler installed

```
$ sudo apt-get install build-essential.
```

Next, git is used to check-out the RTEMS and source-builder source code from the respective repositories, so we must guarantee that is installed. Python-dev contains the header files needed to build Python extensions. And the last command installs the basic GNU tool set which ??.

```
$ sudo apt-get install git
```

```
$ sudo apt-get install python-dev
```

```
$ sudo apt-get build-dep binutils gcc g++ gdb unzip git
```

Besides this, as a final step it is necessary to consult:

```
$ software-properties-gtk
```

This command will open a window on Ubuntu and the Source Code checkbox must be checked.

At this point, the host machine has all the tools ready to download and install the RTEMS toolchain. So, from the command line, we move to */dev/rtems* and clone the source builder source code from the repository (this will put the source code on the folder */dev/rtems/rtems-source-builder*):

```
$ git clone https://github.com/RTEMS/rtems-source-builder.git
```

Finally, we move to the folder that was just cloned */dev/rtems/rtems-source-builder* and perform one last check, to see if the environment was correctly setup.

```
$ source-builder/sb-check
```

This should return the message:

> RTEMS Source Builder environment is OK.

5.2 Source builder toolchain installation

After the environment is correctly set up and we have the source builder code checked out, we can proceed to its build and installation.

```
$. /source-builder/sb-set-builder --log=build-log.txt --prefix=$HOME/dev/rtems/compiler/5 5/rtems-arm
```

We remember that we are working with the version 5 of RTEMS. After this step, which is one of the longest parts of the process, (depending on the computer specs, can take from 10 up to 30 mins) the toolchain is installed under /dev/rtems/compiler/5 and as a final step the bin directory created must be added to our path before we can download and build RTEMS. In linux this is done by editing the *.profile* file and adding PATH=$HOME/dev/rtems/compiler/5/bin:$PATH to it. Or it can be done directly from the terminal, by running:

```
$ export PATH=$HOME/dev/rtems/compiler/5/bin:$PATH
```

5.3 RTEMS configuration/installation

Since we now have the toolchain ready it's time to create the directory where the RTEMS source code will be checked out.

```
$ mkdir /dev/rtems/src
```

```
$ cd src
```

```
$ git clone https://github.com/RTEMS/rtems.git
```

The code will be found on /dev/rtems/src/rtems. The next step is to run the bootstrap script to produce the automatically generated files by autoconf and automake (load the operating system).

After the first clone of the RTEMS repository to run the bootstrap script the following commands are used:

```
$ ./bootstrap -p
```

```
$ ./bootstrap
```

(with the –p option, bootstrap generates preinstall.am files)

Once RTEMS is bootstrapped, it’s time to run the configure script. This specializes de makefile.in files, created by bootstrap, for a specific development host and target.

We start by creating the configure folder. For reasons related with how RTEMS configuration script is made, this folder must have the same name as the target board we are trying to configure, so:

```
$ mkdir realview_pbx_a9_qemu_smp
```

```
$ cd realview_pbx_a9_qemu_smp
```

The options passed to the configure script will determine which tools will be included on the installation of RTEMS. We use:

```
$ $HOME/dev/rtems/src/rtems/configure --target=arm-rtems5 --enable-
rtemsbsp=realview_pbx_a9_qemu_smp --enable-tests=yes --enable-
networking --enable-posix --enable-smp --prefix=$HOME/dev/rtems/bsps/5
```

The command enables the testsuites framework (--enable-tests=yes), the networking (--enable-networking), posix development support (--enable-posix), and the fundamental smp support (--enable-smp). The prefix attribute is where the operating system for the configured target board will be installed. This process makes it easier for developers to work on different target architectures on the same host, as you can easily configure and install different architectures on different directories.

At this point we have the RTEMS operating system configured to run on the arm architecture board, the RealView PBX-A9 baseboard. The only thing left is run the command:

```
$ make
```

inside the dev/rtems/realview_pbx_a9_smp directory and next:

```
$ make install.
```

This will call the Makefile scripts, compile and install the operating system.

The code present on the RTEMS repository has the testsuites folder which contain the set of sample applications already implemented and that can be ran once the installation is complete. At this moment we can use QEMU to run rtems applications by calling:

```
$ qemu-system-arm -no-reboot -nographic -M realview-pbx-a9 -m 256M -
kernel $HOME/dev/rtems/bsps/5-a/arm-
rtems5/realview_pbx_a9_qemu_smp/lib/rtems-5/tests/ticker.exe -smp 2
```

And this is our setup for the development of new test-suites and study of the RTEMS kernel.

(Note: we do not need graphical interfaces to run the samples so that option is disabled. The target board must also be specified and 256mb is the memory allocated for the

applications. What comes after the flag -kernel is the path of the application that we wish to run, followed by -smp X in which X is the number of cores we wish to emulate.

6 Circular Buffer

As mentioned before, the team had the idea to develop a feature that would help verify the correct behaviour of the scheduling mechanisms. It would register all the preemptions, which allowed us to see if both the samples and the scheduler were correct, checking with the design of the samples established before its implementation.

After a deep research and study of RTEMS kernel, mainly the scheduling files, we decided that the appropriated feature would be a circular buffer implemented directly on RTEMS kernel, with the capacity to store 1000 preemptions, when full, it would store the new preemption over the first one that was stored and so on. It runs with any SMP scheduler, the buffer code was developed in the files, *cpustats.h* and *cpustats.c*, both of them stored directly with the other kernel files.

This circular buffer is represented by two structures as it can be seen in figure 13, being the first one, *thread_cpu*, the information related to the preemption, storing the ids of both threads, the moment and in which CPU it occurred. The *Cpu_buffer* is responsible to manage the one thousand instances of *thread_cpu*, its initialization with the flag enabled, that will at the beginning avoid exceptions, and a flag "all" related later to the printing.

```
typedef struct {
        Objects_Id thread_leaving;
        Objects_Id thread_entring;
        rtems_interval ticks;
        uint32_t cpu;
}thread_cpu;

struct Cpu_buffer
{
        thread_cpu history[BUFFER_SIZE];
        uint32_t index;
        uint32_t enabled;
        uint32_t all;
};
```

Figure 13 Circular Buffer structures

The analysis phase of the development of this kernel buffer consisted on, as already mentioned, the studying of the scheduling mechanism in the kernel, but also a comprehension of the initialization manager of RTEMS, due to the necessity to initialize the structure. To ease the development of this buffer we decided to divide it in tree

steps: its creation and initialization, the storing of the preemptions and finally the printing.

6.1 Creation and Initialization

In the first phase, we studied the RTEMS initialization mechanism, due to the need to initialize the buffer as soon as possible to register the firsts preemptions, this initialization would be call by the executive initialization, as it can be seen in figures 14 and 15.

```
void rtems_initialize_executive(void)
{
  const rtems_sysinit_item *item;

  /* Invoke the registered system initialization handlers */
  RTEMS_LINKER_SET_FOREACH( _Sysinit, item ) {
    ( *item->handler )();
  }


  _Init_Sem_Buffer(); <----------

  _System_state_Set( SYSTEM_STATE_UP );

  _SMP_Request_start_multitasking();

  _Thread_Start_multitasking();

  /*******************************************************************
   *******************************************************************
   *******************************************************************
   ******                  APPLICATION RUNS HERE                ******
   ******              THE FUNCTION NEVER RETURNS               ******
   *******************************************************************
   *******************************************************************
   *******************************************************************/
}
```

Figure 14 RTEMS executive initialization

```
static void rtems_initialize_data_structures(void)
{
  /*
   *  Dispatching and interrupts are disabled until the end of the
   *  initialization sequence.  This prevents an inadvertent context
   *  switch before the executive is initialized.
   *
   *  WARNING: Interrupts should have been disabled by the BSP and
   *           are disabled by boot_card().
   */

  /*
   * Initialize any target architecture specific support as early as possible
   */
  _CPU_Initialize();


  _Thread_Dispatch_initialization();

  _ISR_Handler_initialization();

  _Thread_Handler_initialization();

  _Scheduler_Handler_initialization();

  _SMP_Handler_initialize();

  _Buffer_dispath_initialization();

}

RTEMS_LINKER_ROSET( _Sysinit, rtems_sysinit_item );

RTEMS_SYSINIT_ITEM(
  rtems_initialize_data_structures,
  RTEMS_SYSINIT_DATA_STRUCTURES,
  RTEMS_SYSINIT_ORDER_MIDDLE
);
```

Figure 15 RTEMS data structures initialization

First the structure is initialized by the function *_Buffer_dispath_initialization*, allowing it to handle the preemptions and the exceptions if the semaphore is still waiting to be created. The semaphore cannot be created in this moment due to its manager has not been still initialized. The semaphore is used to synchronize the access to the buffer, this synchronization is necessary, once that ***N*** threads can access it at the same time, being ***N*** corresponded to the number of processors configured. It is created by the call of the function *_Init_Sem_Buffer* in a moment where the semaphore manager has already been initialized.

The figures 16 and 17 illustrates the sequence and class diagram of the creation/initialization of the new feature.

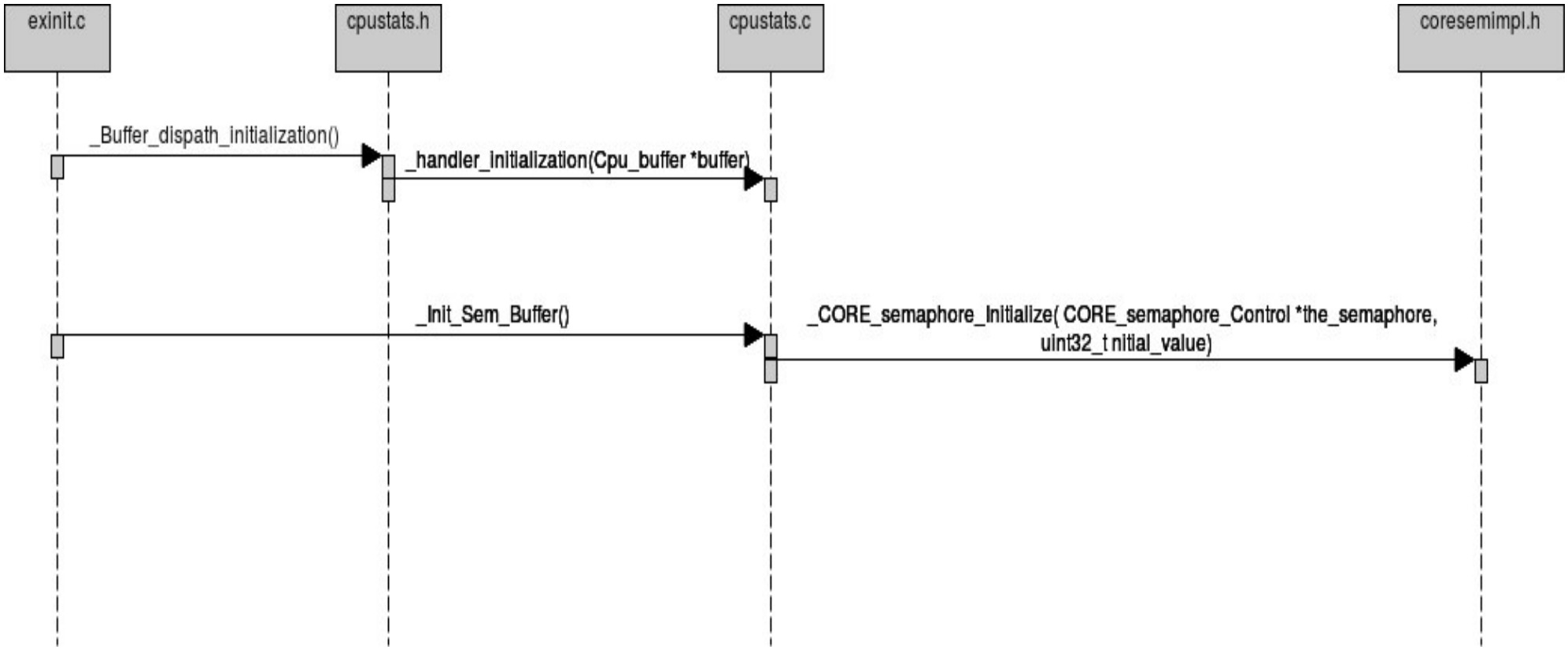


Figure 16 Buffer Initialization Sequence Diagram

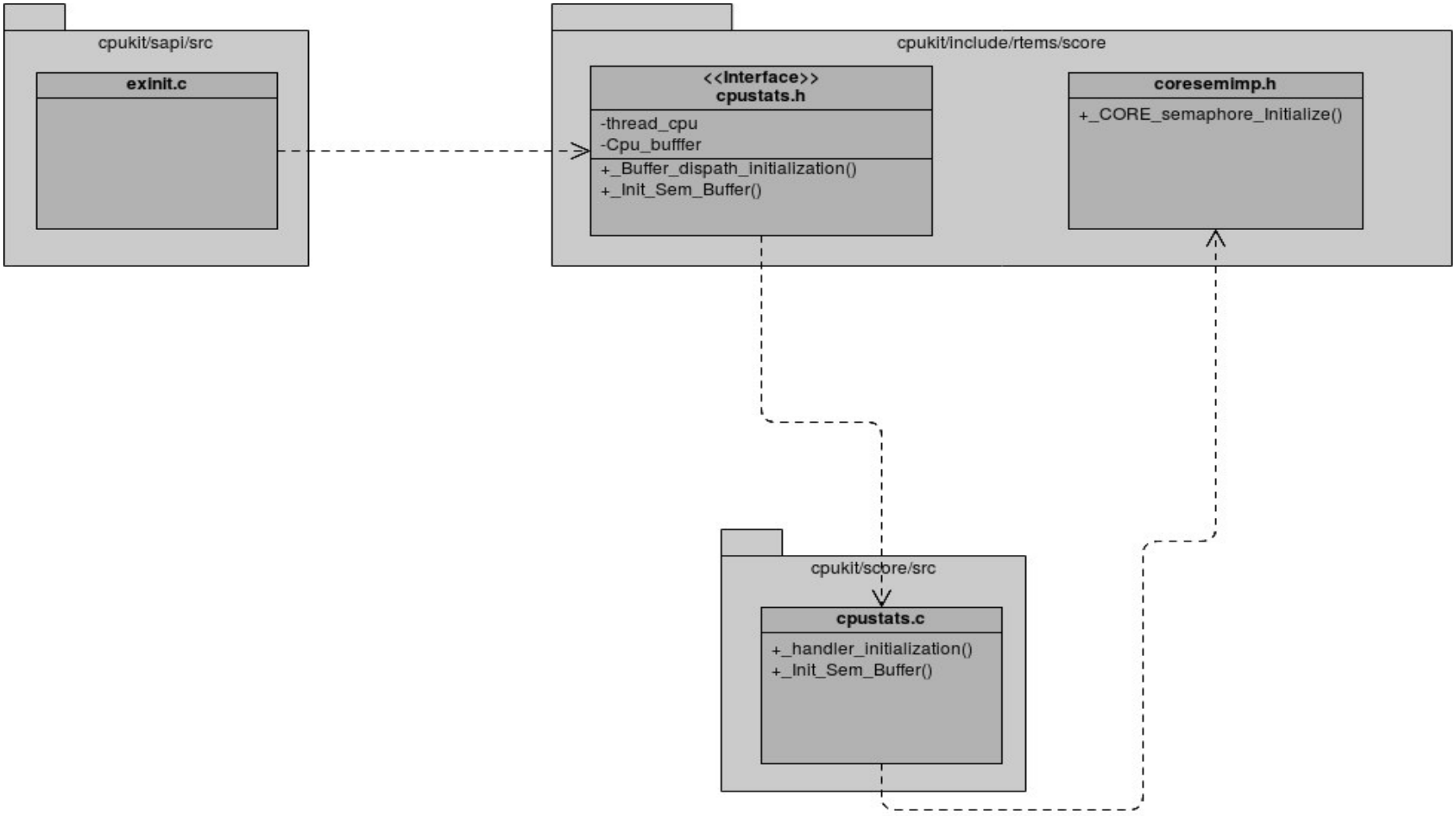


Figure 17 Buffer Initialization Class/File Diagram

6.2 Preemptions

For the second phase, the registration of the preemptions, after a deep research we decided that our code will be called when the operating system is allocating a processor to the entering thread, as it can be seen in figure 18.

```
static inline void _Scheduler_SMP_Allocate_processor(
  Scheduler_Context                *context,
  Scheduler_Node                   *scheduled,
  Scheduler_Node                   *victim,
  Per_CPU_Control                  *victim_cpu,
  Scheduler_SMP_Allocate_processor  allocate_processor
)
{

  Thread_Control *victimthread=_Scheduler_Node_get_user(victim);
  Thread_Control *heirthread=_Scheduler_Node_get_user(scheduled);
  rtems_interval time = rtems_clock_get_ticks_since_boot();

  _AddEvtThread(victimthread,heirthread,time,victim_cpu);

  _Scheduler_SMP_Node_change_state( scheduled, SCHEDULER_SMP_NODE_SCHEDULED );
  ( *allocate_processor )( context, scheduled, victim, victim_cpu );
}
```

Figure 18 Thread allocation

The function *_Scheduler_Node_get_user()* allows to get the *Thread_Control*, that is, all the pertinent information of the entering and leaving threads, followed by *_AddEvtThread()* that sends the information relative to the threads, time and the CPU into the developed code to then store the preemption.

Once the information related to the preemption sent to the buffer code, a semaphore is acquired to guarantee synchronization, the information is stored into the structures that represents the circular buffer, followed by the release of the semaphore.

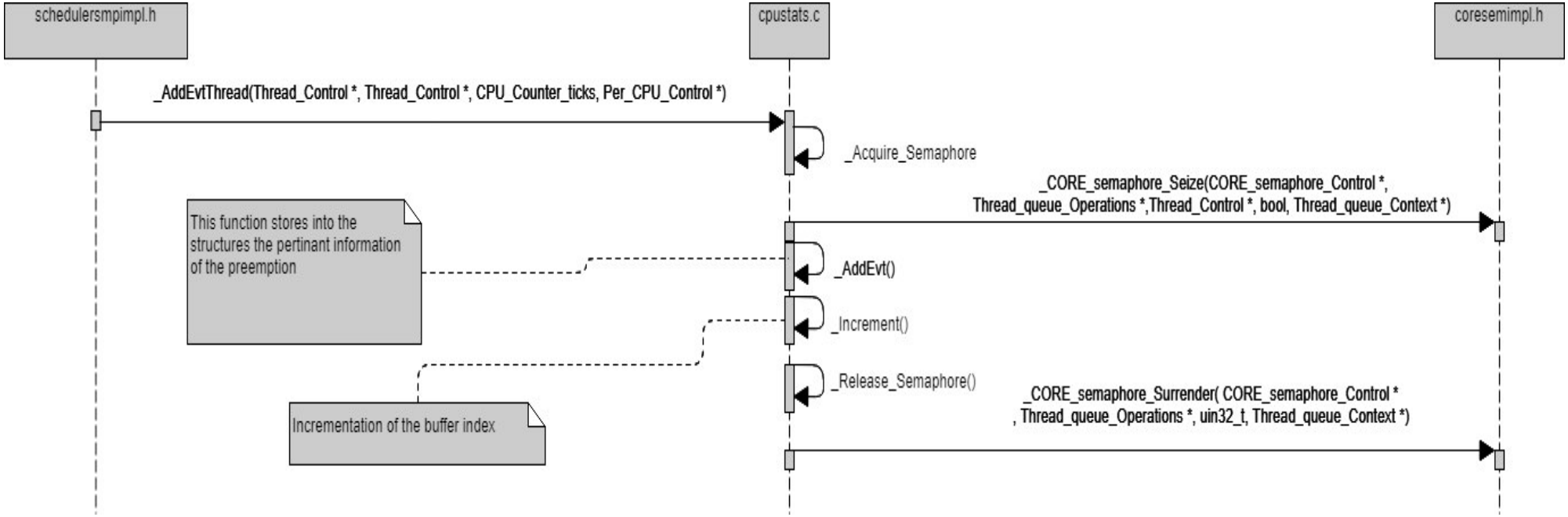


Figure 19 Add event sequence diagram

6.3 Printing

For the final part, the printing, we decided to create a new RTEMS directive, *print_Buffer_CPU_Stats*, that would be called by the RTEMS application as it can be seen in figure 20. The code has been developed in the file *printbuffer.c* and the header *test_support* is used by the RTEMS application to import the buffer printing function.

```
static void Init(rtems_task_argument arg)
{
  TEST_BEGIN();
  Set_up_environment();
  TEST_END();
  print_Buffer_CPU_Stats();
  rtems_test_exit(0);
}
```

Figure 20 new directive to access to the buffer results

When the printing of the buffer is called, a semaphore is acquired to avoid other threads to modify the buffer while we're acceding it to show the preemptions, the semaphore is released at the end of the printing, this sequence and relationship between files and directories is demonstrated in figures 22 and 23.

Figure 21 shows an example of the buffer, with the ids of both threads, the processor and the time in ticks since the boot of the operating system.

```
Index 0 , leaving thread  167837697 , entring thread 151060484  cpu 3 time 0
Index 1 , leaving thread  151060481 , entring thread 167837697  cpu 0 time 1
Index 2 , leaving thread  167837697 , entring thread 151060481  cpu 0 time 1
Index 3 , leaving thread  151060482 , entring thread 167837697  cpu 1 time 1
Index 4 , leaving thread  167837697 , entring thread 151060482  cpu 1 time 1
Index 5 , leaving thread  151060483 , entring thread 167837697  cpu 2 time 1
Index 6 , leaving thread  167837697 , entring thread 151060483  cpu 2 time 1
Index 7 , leaving thread  151060484 , entring thread 167837697  cpu 3 time 1
Index 8 , leaving thread  151060481 , entring thread 167837698  cpu 0 time 1
Index 9 , leaving thread  151060482 , entring thread 167837699  cpu 1 time 1
Index 10 , leaving thread  167837699 , entring thread 167837700  cpu 1 time 71
Index 11 , leaving thread  167837698 , entring thread 151060482  cpu 0 time 101
Index 12 , leaving thread  151060482 , entring thread 167837698  cpu 0 time 101
Index 13 , leaving thread  167837697 , entring thread 151060482  cpu 3 time 101
Index 14 , leaving thread  167837698 , entring thread 151060481  cpu 0 time 117
Index 15 , leaving thread  151060482 , entring thread 167837697  cpu 3 time 122
Index 16 , leaving thread  167837700 , entring thread 167837699  cpu 1 time 123
Index 17 , leaving thread  151060481 , entring thread 167837700  cpu 0 time 123
Index 18 , leaving thread  167837697 , entring thread 151060481  cpu 3 time 123
Index 19 , leaving thread  167837699 , entring thread 151060482  cpu 1 time 124
Index 20 , leaving thread  151060481 , entring thread 167837697  cpu 3 time 124
Index 21 , leaving thread  167837700 , entring thread 151060481  cpu 0 time 124
Index 22 , leaving thread  151060481 , entring thread 167837700  cpu 0 time 124
Index 23 , leaving thread  167837697 , entring thread 151060481  cpu 3 time 124
Index 24 , leaving thread  167837700 , entring thread 151060484  cpu 0 time 124
Index 25 , leaving thread  151060481 , entring thread 167837697  cpu 3 time 124
```

Figure 21 Buffer result

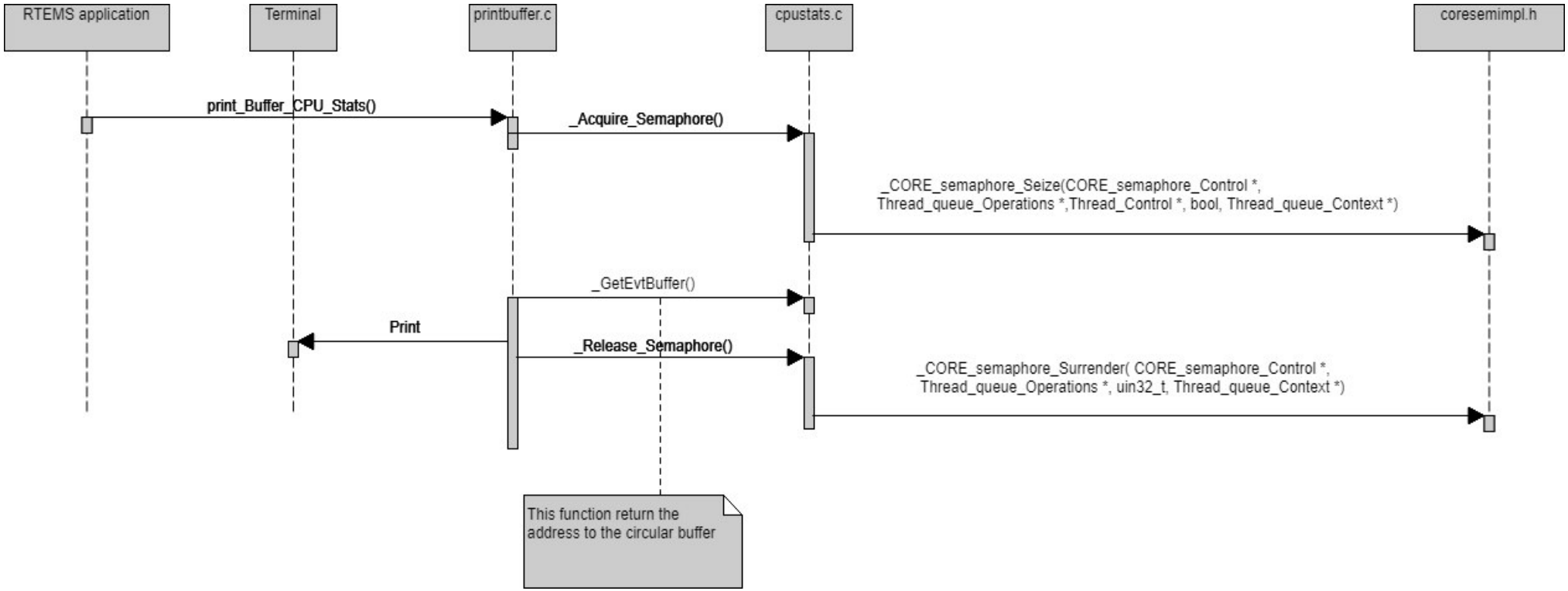


Figure 22 Sequence diagram of the buffer printing

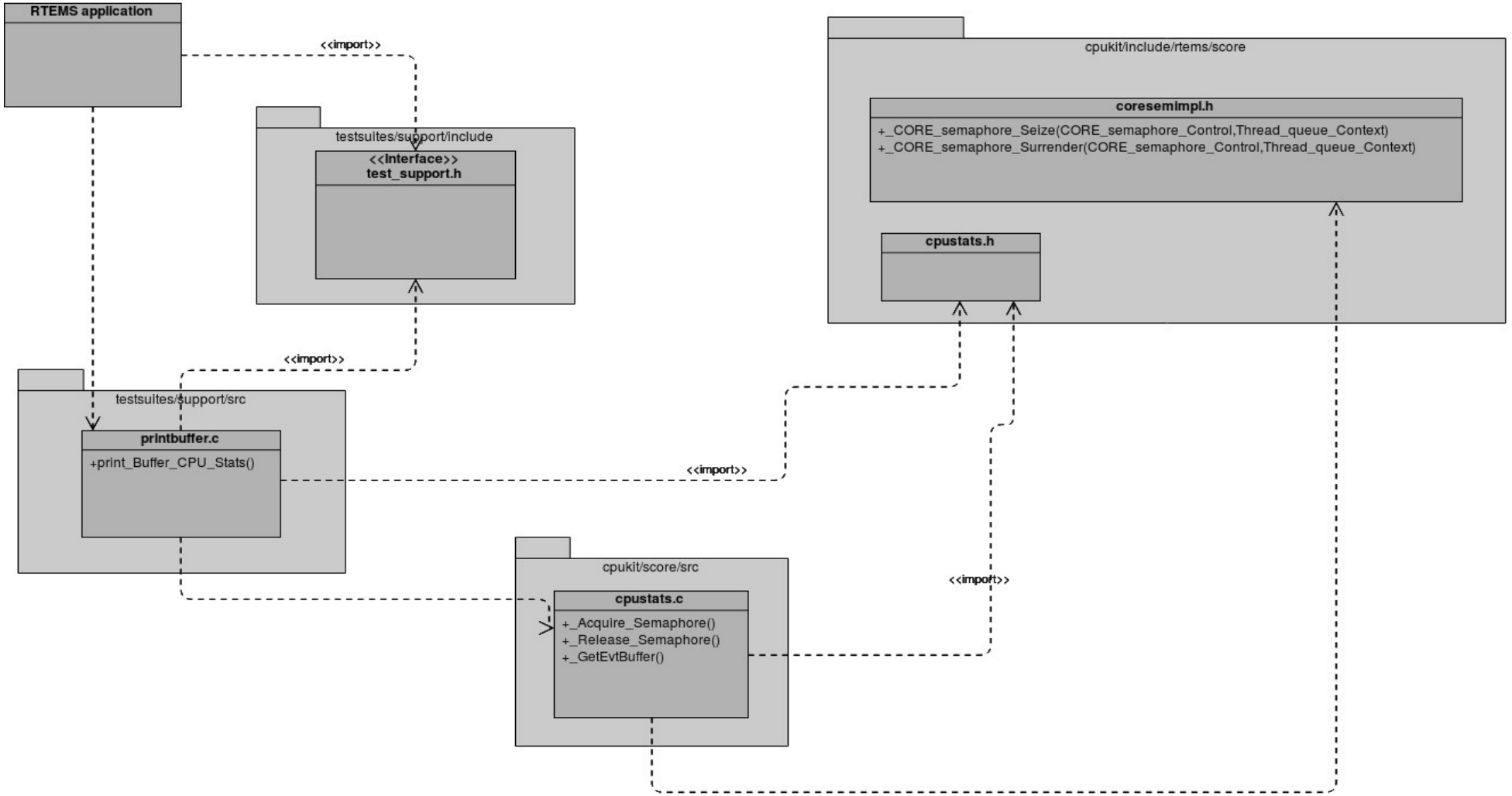


Figure 23 Class/File diagram of the buffer printing

7 RTEMS test suites

From the RTEMS official documentation “Real-time application systems are a special class of computer applications.” since in this case, the correctness of the system depends not only on the results of the computations but also on the time at which they are made. One that is familiar with the design of software systems must be aware, when designing such systems, about the possibility of it being overwhelmed with large numbers of interdependent, asynchronous or cyclical event streams.

The test suites implemented are not bounded by any temporal constraint, unlike the case study implemented later in section 8, that has much more characteristics of a safety-critical application system.

7.2 Scheduling

For the scheduling mechanisms, several new test suites were implemented to verify the correct behaviour of the schedulers. For this verification we developed previously a diagram of the scheduling according with our expectations, that will dictate the development of the RTEMS application.

Then, with the use of the new feature that we developed (circular buffer), we compare the preemptions with our diagram to check if the behaviour occurred has expected.

7.1.1 Smpcistertest01

In this new testsuite we developed a simple RTEMS application to understand, cover and verify the behaviour of the SIMPLE SMP algorithm, a fixed priority scheduler with only one chain for the ready tasks. We configured the scheduler to work on a quad-core platform.

We started by configuring the scheduler, as it can be seen in figure 24, indicating which scheduler we were going to use, to calculate the per-thread overhead introduced. Registering the scheduler in the system via the scheduler table and for last assign the processors to the scheduler.

```
#define CONFIGURE_INIT_TASK_INITIAL_MODES RTEMS_DEFAULT_MODES

#define CONFIGURE_SCHEDULER_SIMPLE_SMP

#include <rtems/scheduler.h>
RTEMS_SCHEDULER_SIMPLE_SMP(a);

#define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
        RTEMS_SCHEDULER_TABLE_SIMPLE_SMP(a,rtems_build_name('S','I','M',' '))

#define CONFIGURE_SCHEDULER_ASSIGNMENT \
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY)

#define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_INIT

#include <rtems/confdefs.h>
```

Figure 24 Simple SMP scheduler configuration

In this sample four tasks were created besides the Init task, that was configured to start the application. This init task is non-preemptable, so, it was always allocated to a processor.

Task	Priority
TA0	10
TA1	10
TA2	15
TA3	5

Table 4 Tasks priorities

In table 4 , we can see the four tasks created, the first three tasks were created at the beginning of the application, so they were directly allocated to the processors, later, TA3 task is created and should preempt TA2, that is the task with lower priority. Figure 25 is the scheduling diagram that represents the expected scheduler behaviour.

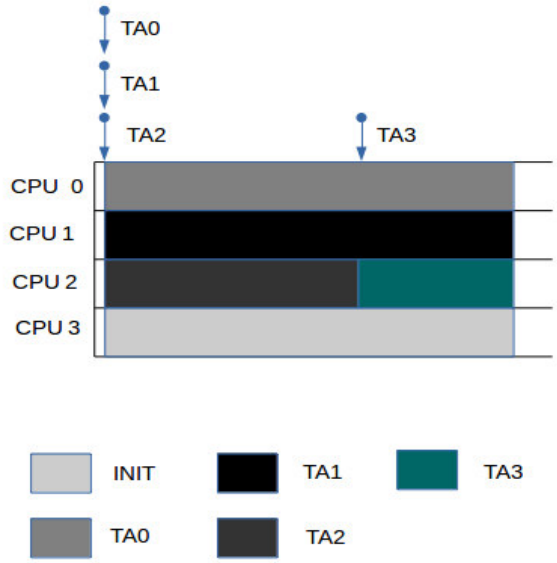


Figure 25 Scheduling diagram SMPCISTERTEST01

We also had to develop the code with the attention that during the execution receive the identifier of the task, to later, when comparing with the result of our new feature, be able to identify which task is which. It can be seen in table 5 the identifiers attributed by RTEMS.

Task	Id
INIT	167837697
TA0	167837698
TA1	1687837699
TA2	167837700
TA3	167837701

Table 5 Tasks Ids SMPCISTERTEST01

After executed the sample, we received the result of the buffer, figure 26, and we verified that the scheduler ran has expected, since the first tree tasks are directly allocated and TA3 preempts TA2, the task with lower priority.

```
Index 0 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 2
Index 1 , leaving thread  151060482 , entering thread 167837699  cpu 1 time 2
Index 2 , leaving thread  151060481 , entering thread 167837700  cpu 0 time 3
Index 3 , leaving thread  167837700 , entering thread 167837701  cpu 0 time 20
```

Figure 26 Buffer result SMPCISTERTEST01

7.1.2 Smpcistertest02

With the sample smpcistertest03 we wanted to test the core affinity of the Arbitrary Processor Affinity Priority SMP scheduler, this scheduler allows the task to run on certain processors, depending on the processor owned by the scheduler instance. So, to start, we configured the scheduler to own all processors configured, as it can be seen in figure 27.

```
#define CONFIGURE_SCHEDULER_PRIORITY_AFFINITY_SMP

#include <rtems/scheduler.h>
RTEMS_SCHEDULER_PRIORITY_AFFINITY_SMP(a,CONFIGURE_MAXIMUM_PRIORITY+1);

#define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
        RTEMS_SCHEDULER_TABLE_PRIORITY_AFFINITY_SMP(a,rtems_build_name('A','F','F',' '))

#define CONFIGURE_SCHEDULER_ASSIGHMENTS \
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
        RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY)

#define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_INIT

#include <rtems/confdefs.h>
```

Figure 27 Arbitrary Processor Affinity Priority SMP scheduler configuration

To change tasks affinity, it was used the diretive *rtems_task_set_affinity,* this directive receives the task id, the size of the *cpu_set_t* and the variable *cpu_set_t,* that indicates with which core there's an affinity. Two functions were used to clear and set bits, CPU_ZERO was used to clear all the bits of *cpu_set_t* and CPU_SET to set the desired bits.

```
cpu_set_t cpu_set;
CPU_ZERO(&cpu_set);
CPU_SET(cpu,&cpu_set);
sc=rtems_task_set_affinity(task_id, sizeof(cpu_set),&cpu_set);
```

Figure 28 Change task affinity

We divided this sample in two phases, in the first one we played with the Init task, moving it from processor to processor. Init begins in the fourth processor, we then move

it to the first, second, third and for last to the fourth again consecutively. Meanwhile, in the second phase we created three tasks with the same priority: TA0, TA1 and TA2.

The first two tasks were created at first, with core affinity to the two first processors, TA0 with the first processor and TA1 with the second one. At this moment, we should have our tasks distributed through the processors as shown in table 6.

Task	Processor
TA0	CPU 0
TA1	CPU 1
Init	CPU 3

Table 6 Tasks allocation

After both tasks were allocated, we created TA2 and set its affinity to CPU 0, CPU1 and CPU 3. As those tree processors were already occupied by tasks with same and/or higher priority and we cleared the bit affinity for the only unoccupied processor *CPU* 2, TA2 was not able to execute at its starting point. To execute TA2 we had to set its priority to a higher one, after the priority changed, the task should preempt TA1 and finally run.

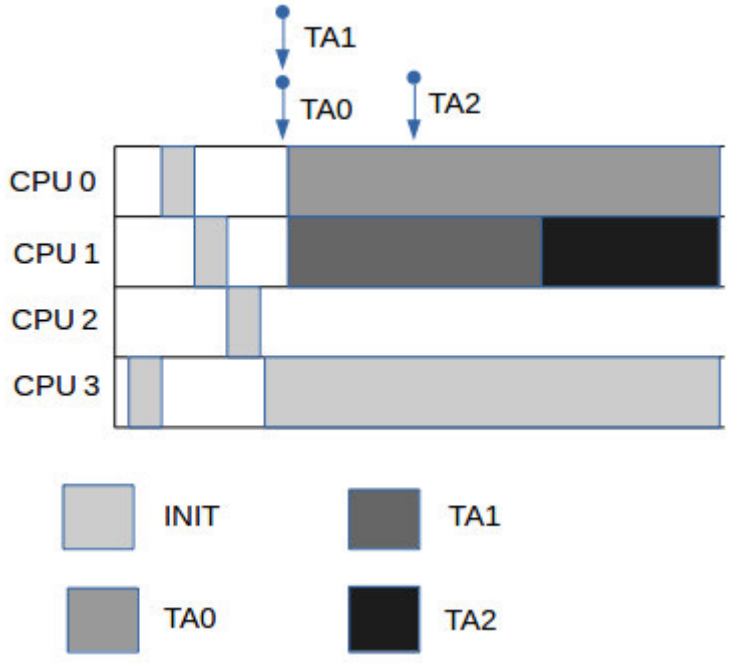


Figure 29 Scheduling diagram SMPCISTERTEST03

In figure 30 we can verify that the init task moved from processor to processor, and we can also check the that TA2 started on tick 31 and do not entered the unoccupied CPU,

it only enters on CPU 1 at time 71, that was when its priority changed and it could preempt TA1.

At the end we compared the expected behavior, figure 31, with the results from our buffer and we were able to conclude that the migrations from the init task, the allocation of TA0 and TA1 in the respective processors and the late preemption of TA1 to TA2 occurred as expected.

Task	Id
Init	167837697
TA0	167837698
TA1	167837699
TA2	167837700

Table 7 Tasks ids SMPCISTERTEST03

```
TA2 created 167837700, tick 31
TA2 seting priority, tick 71
Task 167837700 running on cpu 1, at moment 72
Index 0 , leaving thread  167837697 , entering thread 151060484  cpu 3 time 0
Index 1 , leaving thread  151060481 , entering thread 167837697  cpu 0 time 0
Index 2 , leaving thread  167837697 , entering thread 151060481  cpu 0 time 1
Index 3 , leaving thread  151060482 , entering thread 167837697  cpu 1 time 1
Index 4 , leaving thread  167837697 , entering thread 151060482  cpu 1 time 1
Index 5 , leaving thread  151060483 , entering thread 167837697  cpu 2 time 1
Index 6 , leaving thread  167837697 , entering thread 151060483  cpu 2 time 1
Index 7 , leaving thread  151060484 , entering thread 167837697  cpu 3 time 1
Index 8 , leaving thread  151060481 , entering thread 167837698  cpu 0 time 1
Index 9 , leaving thread  151060482 , entering thread 167837699  cpu 1 time 1
Index 10 , leaving thread  167837699 , entering thread 167837700  cpu 1 time 71
```

Figure 30 Buffer result SMPCISTERTEST03

7.1.3 Smpcistertest03

In our fourth sample, we wanted to work with EDF, a dynamic priority scheduler. Verifying how the scheduler behaves when a background task and a periodic task are executed in the same processor. We did not have the necessity to configure the scheduler since EDF is the default scheduler for SMP configurations.

To create and use periodic tasks we had to resort to the rate monotonic manager, we created the period id using the directive *rtems_rate_monotonic_create,* the task periodicity was later attributed with the use of *rtems_rate_monotonic_period,* this directive initiates the period id with a length of period ticks, if the period id is running, then the calling task will block for the remainder of the period before initiating.

In this sample the init task created both tasks with different priorities, to the periodic *TA0* we attributed a priority of 10 and to the background task *TA1,* 2, and set their affinity to processor *CPU 2,* meaning they only could be executed on this processor.

Since the creation and period attribution must be done by the own periodic task, at the beginning both tasks were saw by the scheduler as two background tasks, and as TA1 had higher priority comparing with TA0, it would enter the CPU and would not free him again.

So, to avoid it, we decided to synchronize the start of both tasks with the use of RTEMS events, we only started TA1 when TA0 would have its periodicity active, allowing TA0 to enter the processor when the periodic task would block waiting for its next period. With that in mind we created the scheduling diagram, figure 32, that would represent the expected behaviour.

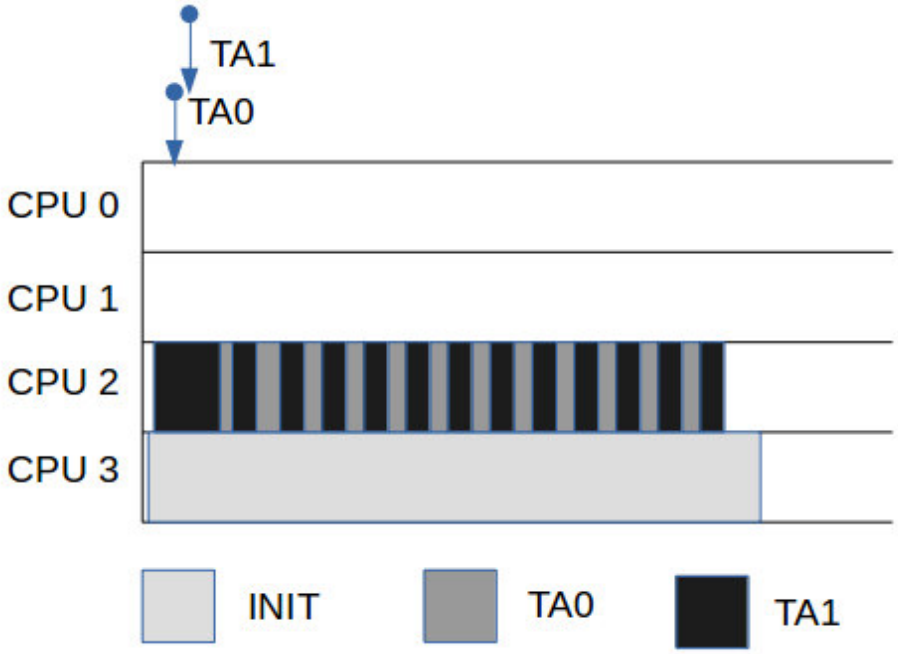


Figure 31 Scheduling diagram SMPCISTERTEST04

After executing the sample, we studied and compared the buffer results. TA1 would always preempt the background task when its period id was unblocked, period of 50 ticks, and when blocked, it was preempted by TA1.

```
TA1 started, tick 12

*** END OF TEST SMPCISTERTEST 4 ***

Index 0 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 0
Index 1 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 34
Index 2 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 52
Index 3 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 83
Index 4 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 102
Index 5 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 133
Index 6 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 152
Index 7 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 182
Index 8 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 202
Index 9 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 232
Index 10 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 252
Index 11 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 282
Index 12 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 302
Index 13 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 332
Index 14 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 352
Index 15 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 382
Index 16 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 402
Index 17 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 432
Index 18 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 452
Index 19 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 482
Index 20 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 502
Index 21 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 532
Index 22 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 552
Index 23 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 562
Index 24 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 562
```

Figure 32 Buffer result SMPCISTERTEST04

7.1.4 Smpcistertest04

This sample is extremely similar to smpcistertest03 sample, described in section 7.1.3, it also tests the dynamic scheduler, but it differs on the proprieties of the tasks. While smpcistertest04 analysed the behaviour of a background task and periodic one fighting for one processor, this sample tests the behaviour of two periodic tasks on the same processor.

Tasks priorities were attributed by the rate monotonic priority assignment policy, I.e. they were inversely to the period, a shorter period has a higher priority. We created two different periods, TA0 with a 50 period ticks and TA1 with a period ticks of 30. Our main aim was to verify the priority assignment behaviour by checking who entered the processor on critical moment.

The scheduling diagram would also be extremely similar to the one from smpcistertest04, in figure 34 we can see the buffer results and confirm that TA1 preempts TA0 when its period became active.

```
Index 0 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 0
Index 1 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 23
Index 2 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 34
Index 3 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 51
Index 4 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 53
Index 5 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 63
Index 6 , leaving thread  167837698 , entering thread 151060483  cpu 2 time 71
Index 7 , leaving thread  151060483 , entering thread 167837699  cpu 2 time 83
Index 8 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 93
Index 9 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 101
Index 10 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 113
Index 11 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 123
Index 12 , leaving thread  167837698 , entering thread 151060483  cpu 2 time 124
Index 13 , leaving thread  151060483 , entering thread 167837699  cpu 2 time 143
Index 14 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 153
Index 15 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 173
Index 16 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 183
Index 17 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 201
Index 18 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 203
Index 19 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 213
Index 20 , leaving thread  167837698 , entering thread 151060483  cpu 2 time 223
Index 21 , leaving thread  151060483 , entering thread 167837699  cpu 2 time 233
Index 22 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 244
Index 23 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 251
Index 24 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 263
Index 25 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 273
Index 26 , leaving thread  167837698 , entering thread 151060483  cpu 2 time 274
Index 27 , leaving thread  151060483 , entering thread 167837699  cpu 2 time 293
Index 28 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 303
Index 29 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 324
Index 30 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 334
Index 31 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 351
Index 32 , leaving thread  167837698 , entering thread 167837699  cpu 2 time 353
Index 33 , leaving thread  167837699 , entering thread 167837698  cpu 2 time 365
Index 34 , leaving thread  167837698 , entering thread 151060483  cpu 2 time 372
Index 35 , leaving thread  151060483 , entering thread 167837699  cpu 2 time 383
Index 36 , leaving thread  167837699 , entering thread 151060483  cpu 2 time 393
Index 37 , leaving thread  151060483 , entering thread 167837698  cpu 2 time 401
```

Figure 33 Buffer result SMPCISTERTEST05

7.1.5 Smpcistertest05

In this sample we only tested the clustered scheduling configuration, defining a processor to each SMP scheduler. We only executed the init task with nop instructions to check if a configuration error occurred.

```
#define CONFIGURE_MAXIMUM_PROCESSORS 4
#define CONFIGURE_MAXIMUM_TASKS 1


#define CONFIGURE_SCHEDULER_EDF_SMP
#define CONFIGURE_SCHEDULER_SIMPLE_SMP
#define CONFIGURE_SCHEDULER_PRIORITY_SMP
#define CONFIGURE_SCHEDULER_PRIORITY_AFFINITY_SMP

#include <rtems/scheduler.h>

RTEMS_SCHEDULER_EDF_SMP(a, CONFIGURE_MAXIMUM_PROCESSORS);
RTEMS_SCHEDULER_SIMPLE_SMP(b);
RTEMS_SCHEDULER_PRIORITY_SMP(c, CONFIGURE_MAXIMUM_PRIORITY +1);
RTEMS_SCHEDULER_PRIORITY_AFFINITY_SMP(d,CONFIGURE_MAXIMUM_PRIORITY +1);

#define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
  RTEMS_SCHEDULER_TABLE_EDF_SMP(a, rtems_build_name('E', 'D', 'F', ' ')),\
  RTEMS_SCHEDULER_TABLE_SIMPLE_SMP(b, rtems_build_name('S','I','M',' ')),\
  RTEMS_SCHEDULER_TABLE_PRIORITY_SMP(c, rtems_build_name('P','R','I',' ')),\
  RTEMS_SCHEDULER_TABLE_PRIORITY_AFFINITY_SMP(d, rtems_build_name('A','F','F',' '))

#define CONFIGURE_SCHEDULER_ASSIGNMENTS \
  RTEMS_SCHEDULER_ASSIGN(0,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
  RTEMS_SCHEDULER_ASSIGN(1,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
  RTEMS_SCHEDULER_ASSIGN(2,RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
  RTEMS_SCHEDULER_ASSIGN(3, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY)

#define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION

#define CONFIGURE_RTEMS_INIT_TASKS_TABLE

#define CONFIGURE_INIT


#include <rtems/confdefs.h>
```

Figure 34 Clustered scheduling configuration

7.2 Communication and Synchronization

The set of test suites present in the repository already contemplated some synchronization and communication mechanisms. The subsection 7.2.1 presents the names and a brief description of what the applications aim to test.

Images 36, 37 and 38 present the structure followed to develop the synchronization test-suites (identical structure to the scheduling samples).

```
193  /*Init function*/
194  rtems_task Init(rtems_task_argument arg)
195  {
196    test_context *ctx = &test_instance;
197
198    TEST_BEGIN();
199    locked_print_initialize();
200
201    for (int i = 0; i < 3; i++)
202    {
203      TaskRan[i] == false;
204    }
205    locked_printf("\n************ ********* ************\n");
206    PrintTaskInfo("Init"); //imprime as informações da task Init
207    test(ctx);
208    locked_printf("************ ********* ************\n\n");
209    TEST_END();
210    rtems_test_exit(0);
211  }
```

Figure 35 Example of Init task

```
104  /*
105  * ================================================== Init Function ==================
106  * ================================================================================
107  */
108  static void test(test_context *ctx)
109  {
110    rtems_status_code sc;
111    bool go = true;
112
113    //_CREATE barrier that oneps automatically when third task arrives
114    sc = rtems_barrier_create(
115        BAR_NAME,
116        RTEMS_BARRIER_AUTOMATIC_RELEASE,
117        3,
118        &ctx->bar_id);
119    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
120
121    //_CREATE Task_1 - Global
122    sc = rtems_task_create(
123        rtems_build_name('T', 'A', '1', ' '),
124        TASKS_PRIO,
125        RTEMS_MINIMUM_STACK_SIZE,
126        RTEMS_DEFAULT_MODES,
127        RTEMS_GLOBAL,
128        &ctx->task[0]);
129    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
130
131    // CREATE Task_2 - Local
```

Figure 36 Example of Init task - 2

```
46  /*Task 1*/
47  rtems_task Task_1(rtems_task_argument arg)
48  {
49    test_context *ctx = (test_context *)arg;
50    rtems_status_code sc;
51    rtems_task_priority priority;
52
53    /* Show that this task is waiting on the barrier, running on cpu X */
54    PrintTaskInfo("TA1: Waiting at the barrier");
55    sc = rtems_barrier_wait(
56        ctx->bar_id,
57        RTEMS_WAIT);
58    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
59
60    PrintTaskInfo("TA1 END.");
61    TaskRan[0] = true;
62    rtems_task_suspend(RTEMS_SELF);
63    rtems_test_assert(0);
64  }
65
66  /*Task 2*/
67  rtems_task Task_2(rtems_task_argument arg)
68  {
69    rtems_status_code sc;
70    test_context *ctx = (test_context *)arg;
71
72    /* Show that this task is waiting on the barrier, running on cpu X */
73    PrintTaskInfo("TA2: Waiting at the barrier");
```

Figure 37 Example of RTEMS tasks

The Init task, which acts as a “main” function starts all other tasks that will interact in the system.

We can have the code related with the different tasks on different *.c* files but to follow the practises of the official samples on the RTEMS repository each of the scheduler and synchronization samples where created on the same file (the Init function and task routines are written on the same file). The *system.h* is the header file where the configurations are stored.

```
);

rtems_task Task_3(
rtems_task_argument arg
);

/****************************** configuration data ************************
/**********************************************************************
#define CONFIGURE_MAXIMUM_SEMAPHORES 2
#define CONFIGURE_MAXIMUM_BARRIERS 1

//#define CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_SIMPLE_CONSOLE_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_MICROSECONDS_PER_TICK   1000 /* 1 millisecond */

#define CONFIGURE_MAXIMUM_PRIORITY 255
#define CONFIGURE_MAXIMUM_PROCESSORS 4
#define CONFIGURE_MAXIMUM_TASKS 4

//STEP 1 tell the system what scheduler algorithm to use

#define CONFIGURE_SCHEDULER_PRIORITY_SMP
#include <rtems/scheduler.h>
//STEP 2 - configure THE SCHEDULER INSTANCES
RTEMS_SCHEDULER_PRIORITY_SMP(a, CONFIGURE_MAXIMUM_PRIORITY + 1);
RTEMS_SCHEDULER_PRIORITY_SMP(b, CONFIGURE_MAXIMUM_PRIORITY + 1);

//STEP 3
#define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
RTEMS_SCHEDULER_TABLE_PRIORITY_SMP( \
  a, \
   SCHED_A \
), \
RTEMS_SCHEDULER_TABLE_PRIORITY_SMP( \
  b, \
```

Figure 38 Example of a configuration file

(Note: The file is not fully printed due to the lack of necessity in occupying more space already. The point, as referred earlier, was to give a broad idea on the structure of this test-suites)

SMP test suites already Implemented

These samples can be found on the official RTEMS github page. Our team had nothing to do with the development of such. Since they already test some synchronization mechanisms, it was important to make reference to them.

Semaphores

test name: **smppsxmutex01**
directives tested: pthread_mutex_lock()
objectives:

- Ensure that priority ceiling mutexes work only in their dedicated scheduler instance.

test name: **smpmutex01**
directives tested:

- Thread_queue_Priority_do_enqueue()
- Thread_queue_Priority_do_extract()
- Thread_queue_Priority_first()

objectives:

- Ensure that the thread queue priority discipline enforces FIFO fairness among the highest priority thread of each scheduler instance.

Locking protocols: MrsP

test name: **smpmrsp01**
directives tested:

- _MRSP_Initialize()
- _MRSP_Obtain()
- _MRSP_Timeout()
- _MRSP_Release()
- _MRSP_Get_ceiling_priority()
- _MRSP_Set_ceiling_priority()

objectives:

- Ensure that rtems_semaphore_flush() returns an error status for MrsP semaphores.
- Ensure that rtems_semaphore_create() for an initially locked semaphore returns an error status for MrsP semaphores.

- Ensure that a nested obtain rtems_semaphore_obtain() returns an error status for MrsP semaphores.
- Ensure that a rtems_semaphore_obtain() leading to a deadlock returns an error status for MrsP semaphores.
- Ensure that it is possible to obtain multiple MrsP semaphores.
- Ensure that a timeout on MrsP semaphores works.
- Ensure that heavy usage of multiple MrsP semaphores works.

test name: **smpmutex02**
directives tested: rtems_semaphore_obtain()
rtems_semaphore_release()
objectives:

- Ensure that arbitrary mutex obtain sequences carried out by multiple threads on multiple processors work.

Barriers: *No tests implemented*

Message Queues

test name: **smpipi01**
directives tested: _SMP_Send_message()
objectives:

- Ensure that SMP message delivery works in the context of an SMP message handler.
- Ensure that a flood of inter-processor interrupts works as expected.

Events: *No tests implemented*

Signals

test name: **smpsignal01**
directives tested: **rtems_signal_catch()**
rtems_signal_send()
objectives:

- Ensure that signal handlers are called with interrupts enabled.
- Ensure that Classic Signals work on SMP.

We can see above that, smpmutex02 already tests, in multiple processors, the directive used to obtain and release semaphores. And barriers and events have no sample application implemented so this must be adressed. Focusing on the locking protocols, all the directives that are implemented on the kernel for MrsP are tested in smpmrsp01. Connected with the locking protocols is the helping hand protocol, that was discussed on section 4, and involves the helping mechanism implemented to deal with priority inversion. Since there was no sample regarding the latter, we created it.

After gathering the information about the directives that were already tested, what those that were not, we started the design of our own.

7.2 SMP test-suites to Implement

The following subsections presents the details of the test-suites implemented to target the synchronization and communication mechanisms of RTEMS.
These tests focus mainly on the Semaphore Manager, Barrier Manager and Message Manager.

7.2.1 Semaphores (with MrsP and OMiP protocol)

test name: smptestdev01
directives tested: -
objectives:

- Ensure that the task in ownership of the mutex migrates to another scheduler instance in case it is preempted. (helping protocol)

The purpose of this sample was to test the helping mechanism the locking protocols provide when a task that currently owns a mutex is preempted (discussed on section 4.). This simulates an environment where the system has tasks running, or ready to run that can preempt the locked-holding task.

The application was developed by configuring two instances of the Deterministic Priority SMP Scheduler. (clustered scheduling.) We'll call them SCHED_A and SCHED_B. SCHED_A is responsible for scheduling tasks on CPU 1 and 2, and SCHED_B on CPU 2 and 3.

- On SCHED_A we have the Task Init, Task 1 (T1) and Task 2 (T2) scheduled. (Preemption is disabled for the Init task.)
-
- On SCHED_B we have Task 3 (T3) that has the highest priority of all the tasks in the system and will be contending to obtain the mutex.

Task	Priority
	Priority Ceiling - 9
Init	5
Task 3	10
Task 2	8

Task 1	20

Table 8 Tasks Priorities SMPTESTDEV01

Table 8 gives a visual representation on how the tasks are running on CPU 1 and 3.
T3 arrives at the processor when T1 is already running the critical section of the code, so T3 will have to wait.
At time B, T1 is preempted by T2, which is not a contendant to obtain the semaphore and has a higher priority than the priority ceiling. At this point, T3 gives the possibility of the job of T1 to be ran on the CPU 3, where T3 is on, on another scheduler instance.
At instance C, T1 finished running the critical section, and so T3 can start its execution, which otherwise would only begin sometime later.
T2 finishes on instant D, where T1 returns to the processor. At this moment, T1 starts its execution right after the critical section.

The configuration information on system.h is visible on the following images.

```
10  #include "tmacros.h"
11
12  /*
13  * Variable configurations.
14  */
15  #define SCHED_A rtems_build_name('A', ' ', ' ', ' ')
16  #define SCHED_B rtems_build_name('B', ' ', ' ', ' ')
17
18  /*
19  * Functions declaration.
20  */
21
22  void PrintTaskInfo(
23  const char *task_name
```

Figure 39 configuration file (1)

The names of the different schedulers are defined here (image 40). In this case we will configure two.

```
42  /****************************** configuration information ********************
43  /*************************************************************************************
44  #define CONFIGURE_MAXIMUM_SEMAPHORES 3
45  #define CONFIGURE_MAXIMUM_MRSP_SEMAPHORES 1
46
47  #define CONFIGURE_APPLICATION_NEEDS_SIMPLE_CONSOLE_DRIVER
48  #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
49  #define CONFIGURE_MICROSECONDS_PER_TICK   1000 /* 1 millisecond */
50
51  #define T1PRIO 20
52  #define T2PRIO 8
53  #define T3PRIO 10
54  #define PRIORITY_CEILING 9
55  #define CONFIGURE_MAXIMUM_TASKS 4
56
57  #define CONFIGURE_MAXIMUM_PRIORITY 255
58  #define CONFIGURE_MAXIMUM_PROCESSORS 4
59
60  #define CONFIGURE_SCHEDULER_PRIORITY_SMP
61  #include <rtems/scheduler.h>
62  //STEP 2 - configure THE SCHEDULER INSTANCES
63  RTEMS_SCHEDULER_PRIORITY_SMP(a, CONFIGURE_MAXIMUM_PRIORITY + 1);
64  RTEMS_SCHEDULER_PRIORITY_SMP(b, CONFIGURE_MAXIMUM_PRIORITY + 1);
65
66  //STEP 3
67  #define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
68  RTEMS_SCHEDULER_TABLE_PRIORITY_SMP( \
```

Figure 40 configuration file (2)

Everything on RTEMS must be defined. On this case we used an MRSP semaphore first so we must declare how many we will use (red arrow on figure 40). Also note the priorities defined, as mentioned before.

```
66  //STEP 3
67  #define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
68  RTEMS_SCHEDULER_TABLE_PRIORITY_SMP( \
69    a, \
70    SCHED_A \
71  ), \
72  RTEMS_SCHEDULER_TABLE_PRIORITY_SMP( \
73    b, \
74    SCHED_B \
75  )
76
77  //STEP 4 Scheduler to processor assignment
78  #define CONFIGURE_SCHEDULER_ASSIGNMENTS \
79  RTEMS_SCHEDULER_ASSIGN(0, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
80  RTEMS_SCHEDULER_ASSIGN(0, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
81  RTEMS_SCHEDULER_ASSIGN(1, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY), \
82  RTEMS_SCHEDULER_ASSIGN(1, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY)
83
84  #define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION
85
86  #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
87
88  //#define CONFIGURE_INIT_TASK_STACK_SIZE \
89    (3 * CONFIGURE_MINIMUM_TASK_STACK_SIZE)
90
91  #include <rtems/confdefs.h>
```

Figure 41 configuration file (3)

The scheduler assignments are made on the following manner (as seen on figure 42). The numbers 0 and 1 correspond to the entrances on the table designated RTEMS_SCHEDULER_TABLE_PRIORITY_SMP. And each entrance on the table named CONFIGURE_SCHEDULER_ASSIGNMENTS correspond to one processor in the system. Since we have 4 processors we will have 4 lines on the table. (typical example of clustered scheduling configuration).

The challenge here was to make the tasks enter at the right times to simulate a problem like the one described. Task3 must be scheduled to enter right after Task1 obtained the semaphore.

```
71  //task 1 ==================================================
72  rtems_task Task_1(
73      rtems_task_argument arg)
74  {
75    test_context *ctx = (test_context *)arg;
76    rtems_status_code sc;
77
78    //Obtain the semaphore - CRITICAL SECTION
79    sc = rtems_semaphore_obtain(ctx->semaphore_id, RTEMS_WAIT, RTEMS_NO_TIMEOUT);
80    printf("ta1_sem_obtain_status: %s\n", rtems_status_text(sc));
81    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
82
83    printf("* T1 Counting . . . 0%\n");
84    for(int i = 0; i < 2000000; i++){
85      //simulate work
86    }
87    printf("* T1 Counting . . . 100%\n");
88    ctx->t2_flag = true;
89    //Release the semaphore
90    sc = rtems_semaphore_release(ctx->semaphore_id);
91    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
92
93    TaskRan[0] = true;
94    rtems_task_suspend(RTEMS_SELF);
95    rtems_test_assert(0);
96  }
```

Figure 42 TASK1 - code

So, while Task1 is running the *for* loop seen on figure 43, Init will start Task3 and Task2. Task3 will start by calling rtems_semaphore_obtain() directive to start its execution on the critical section and will have to wait. Task2 will preempt Task1, sending it to another scheduler instance (thanks to the helping protocol).
The output supports this expected behavior.

To test the OMiP protocol the only changes made to this sample test are present on the rtems_semaphore_create() directive.

```
150   //CRIAR O MrsP SEMAPHORE
151   sc = rtems_semaphore_create(
152       rtems_build_name('M', 'R', 'S', 'P'),
153       1,
154       RTEMS_BINARY_SEMAPHORE  | RTEMS_MULTIPROCESSOR_RESOURCE_SHARING,
155       PRIORITY_CEILING,
156       &ctx->semaphore_id
157       );
158   //printf("create_semaphore_status: %s\n", rtems_status_text(sc) );
159   rtems_test_assert(sc == RTEMS_SUCCESSFUL);
```

Figure 43 Creating of an MrsP semaphore

Instead of using the attribute set:
RTEMS_BINARY_SEMPAHORE | RTEMS_MULTIPROCESSOR_RESOURCE_SHARING
we used:
RTEMS_BINARY_SEMPAHORE | RTEMS_INHERIT PRIORITY

The OMiP protocol is not at all addressed on the semaphore manager of the RTEMS documentation. Also, on the entire documentation there is not one example of this protocol in use even though it is clear it is implemented. The way RTEMS makes available the two different protocols and even the table of available attribute sets that is present on the semaphore manager web-page are quite misleading, here's why:

Locking protocol	Flag
Priority Ceiling (uniprocessor)	RTEMS_PRIORITY_CEILING
Priority Inheritance (uniprocessor)	RTEMS_INHERIT_PRIORITY
MrsP (multiprocessing)	RTEMS_MULTIPROCESSOR_RESOURCE_SHARING
OMiP (multiprocessing)	RTEMS_INHERIT_PRIORITY

Table 9 Locking Protocols

So, the RTEMS user is left to guess that the same flag used for uniprocessor configurations is re-used on the multiprocessor ones. This problem was brought up on an e-mail sent to the RTEMS users mailing lists and Sebastian Huber from embedded brains acknowledged the documentations should be made clearer.

7.2.2 Barriers

test name: smptestdev02
directives tested:

- rtems_barrier_create()
- rtems_barrier_wait()

objectives:

- Ensure that classic barrier configuration works properly on SMP.

design/Implementation:
This small sample represents a classic RTEMS configuration of a barrier that automatically opens when a certain number of tasks are blocked at it. We designed a simple application with 3 tasks that will call rtems_barrier_wait() directive to wait at the barrier.

```
42  /*************************** configuration information ***********************/
43  /*****************************************************************************/
44  #define CONFIGURE_MAXIMUM_SEMAPHORES 2
45  #define CONFIGURE_MAXIMUM_BARRIERS 1
46
```

Figure 44 Semaphores header file

The configuration settings on the header file must specify the maximum number of barriers used (1). There are 2 semaphores declared, one is used for the

locked_print_f() function, in order to synchronize the console writing between tasks, and the other is the barrier.

```
//_CREATE barreira que abre automaticamente quando a 3ª task chega
sc = rtems_barrier_create(
    BAR_NAME,
    RTEMS_BARRIER_AUTOMATIC_RELEASE,
    3,
    &ctx->bar_id);
rtems_test_assert(sc == RTEMS_SUCCESSFUL);
```

Figure 45 Barrier creation

It is on the Init task that the barrier is created, right before the creation of the other tasks, that will wait at it. RTEMS directive rtems_barrier_wait() receives the configuration parameters. In this case, we will have a barrier that opens automatically when the third task arrives.

The Barrier Manager section of the RTEMS official documentation provides a good description on the configuration details of said mechanism.

```
46  /*Task 1*/
47  rtems_task Task_1(rtems_task_argument arg)
48  {
49    test_context *ctx = (test_context *)arg;
50    rtems_status_code sc;
51    rtems_task_priority priority;
52
53    /* Show that this task is waiting on the barrier, running on cpu X */
54    PrintTaskInfo("TA1: Waiting at the barrier");
55    sc = rtems_barrier_wait(
56        ctx->bar_id,
57        RTEMS_WAIT);
58    rtems_test_assert(sc == RTEMS_SUCCESSFUL);
59
60    PrintTaskInfo("TA1 END.");
61    TaskRan[0] = true;
62    rtems_task_suspend(RTEMS_SELF);
63    rtems_test_assert(0);
64  }
65
```

Figure 46 Task 1 code

The three tasks have roughly the same code. The function *PrintTaskInfo(const char *task_name*) prints the information on the console. Each task prints the ID before calling *rtems_barrier_wait*(), and after being released.

```
*** BEGIN OF TEST SMP_TEST_DEV 02 ***
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)

************ ********* ************
* CPU 3 running task Init
id 167837698
id 167837699
id 167837700
* CPU 2 running task TA1: Waiting at the barrier
* CPU 1 running task TA2: Waiting at the barrier
* CPU 2 running task TA3: Waiting at the barrier
* CPU 2 running task TA3 END.
* CPU 1 running task TA2 END.
* CPU 1 running task TA1 END.
************ ********* ************


*** END OF TEST SMP_TEST_DEV 02 ***
```

Figure 47 Barrier sample - output

It is visible on the output on image 48, each task printing they have arrived at the barrier, and only after the three printed, they all finalize. It is important to note the barrier implementation on RTEMS offers only a FIFO blocking order on the waiting queue. This is because the tasks are released as a set, and it is expected for them to compete for the processors based upon their priority levels.

7.2.3 Message Queues

test name: smptestdev03

directives tested:

- rtems_message_queue_create()
- rtems_message_queue_receive()
- rtems_message_queue_send()

objectives:

- Ensure message queue synchronization mechanisms work correctly on an SMP configuration. (FIFO and priority ordering)

design/Implementation:
The configuration part of this sample contemplates the definition of how many message queues the application will require and the size of those queues.

```
67    #define CONFIGURE_MAXIMUM_MESSAGE_QUEUES 2
68    #define CONFIGURE_MESSAGE_BUFFER_MEMORY \
69            CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE(50, sizeof(task_message))
70
```

Figure 48 Message queues configuration

The macros seen on figure 49 configure two message queues (one with FIFO ordering, other with priority order) that can hold 50 messages each.

The size of one message is equal to the size of the Task_message variable, which is a structure that holds a char array of size 40. This means the application will require 2 * 50 * 40 = 4000 bytes of memory to be allocated for this message queue configuration. The message queues are identified by their ids, that are returned when Init calls the *rtems_message_queue_create()* directive.

```
178     //_CREATE message with fifo order of waiting tasks
179     sc = rtems_message_queue_create(
180         MSGQ1_NAME,
181         MSG_COUNT,
182         sizeof(task_message),
183         RTEMS_FIFO,
184         &ctx->msq1_id);
185     rtems_test_assert(sc == RTEMS_SUCCESSFUL);
186
187     //_CREATE message with priority order of waiting tasks
188     sc = rtems_message_queue_create(
189         MSGQ2_NAME,
190         MSG_COUNT,
191         sizeof(task_message),
192         RTEMS_PRIORITY,
193         &ctx->msq2_id
194         );
195     rtems_test_assert(sc == RTEMS_SUCCESSFUL);
```

Figure 49 Message queue creation

There are 4 intervening tasks. (counting with Init) Task3 is the “speaker”. Task1 and Task2 are “listeners”. This means Task3 will be the only one writing on the message queues. Task1 is the higher priority task between the two:

- Priority of Task1: 10
- Priority of Task2: 20

We create two message queues. On MSG_QUEUE_1 tasks wait on a first-in-first-out manner. On MSG_QUEUE_2, The first task serviced is the higher priority one, among the tasks waiting.

Task3 will be sending messages where Task1 and 2 are waiting to proceed.

```
/*Task 1 - Listener  ################################*/
rtems_task Task_1(rtems_task_argument arg)
{
  test_context *ctx = (test_context *)arg;
  size_t msg_size = sizeof(task_message);
  rtems_status_code sc;

  /*Waiting on Task3 message on msq2*/
  sc = rtems_message_queue_receive(
    ctx->msq2_id,
    (void *)ctx->msg01.body,
    &msg_size,
    RTEMS_WAIT,
    RTEMS_NO_TIMEOUT
  );
  rtems_test_assert(sc == RTEMS_SUCCESSFUL);

  locked_printf("* TASK1: (should be the first to report.) \n%s\n\n", ctx->msg01.body);

/*Waiting on Task3 message now msq1*/
  sc = rtems_message_queue_receive(
    ctx->msq1_id,
    (void *)ctx->msg01.body,
    &msg_size,
    RTEMS_WAIT,
    RTEMS_NO_TIMEOUT
  );
  rtems_test_assert(sc == RTEMS_SUCCESSFUL);

  locked_printf("* TASK1: (should be the first to report.) \n \"%s\"\n\n", ctx->msg01.body);

  TaskRan[0] = true;
  rtems_task_suspend(RTEMS_SELF);
  rtems_test_assert(0);
}
```

Figure 50 Task1 - Message Queue

When the *directive rtems_message_queue_receive()* is called, we specify the behavior of the task when waiting for a message. This is visible on figure 40. Since we defined RTEMS_WAIT and RTEMS_NO_TIMEOUT the tasks will wait forever on a message to arrive on the message queue specified by ctx->msq1_id (a structure where the variable ID's are stored).

Task1 and 2 have approximately the same code. They execute with the same behavior. First wait for a message on MSG_QUEUE_2 (priority-based waiting), print the message received and then wait on MSG_QUEUE_1.

Task3 must send 2 messages for each message queue as each message will unblock each of the tasks waiting.

Since Task1 has a higher priority than Task2, it will unblock first, reaching the second call of *rtems_message_queue_receive()* first too. So, at end we will see that Task1 always prints the messages first (as it should).

```
*** BEGIN OF TEST SMP_TEST_DEV 03 ***
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)

************ ********* ************
* CPU 3 running task Init
id 167837698
id 167837699
id 167837700
* TASK1: (should be the first to report.)
Written on MSQ2. by TASK 03.
* TASK2:
Written on MSQ2. by TASK 03.

* TASK1: (should be the first to report.)
 "Written on MSQ1. by TASK 03."
* TASK2:
Written on MSQ1. by TASK 03.

************ ********* ************

*** END OF TEST SMP_TEST_DEV 03 ***
```

Figure 51 MSQ sample - output

7.2.4 Events

test name: smptestdev04

directives tested:

- *rtems_event_send()*
- *rtems_event_receive()*

objectives:

- Ensure that classic events work properly on an SMP configuration
- Ensure that RTEMS_EVENT_ALL works.

design/Implementation:

In SMP events can be pretty much kept without a complete re-design of its implementation, and they are developed to be used as simple synchronization mechanisms.

In RTEMS tasks can wait on more than one event simultaneously and event flags are used to manage event sets. The set of valid events goes from the macro RTEMS_EVENT_0 to RTEMS_EVENT_31.

On this sample, we have 3 tasks that will be using events, the following design was implemented:

- Task 3 will be sending events to Task 1 and 2.
- Task 1 is waiting on events 6 and 7.
- Task 2 is waiting on event 7.

In order to build the event set, on Task 3 we call the directive rtems_event_send() and pass the configurations by parameter.

```
86      /*Send EVENT_6 to Task1 who is waiting on EVENT_6 AND EVENT_7*/
87      sc = rtems_event_send(
88          ctx->task[0],
89          RTEMS_EVENT_6);
90      rtems_test_assert(sc == RTEMS_SUCCESSFUL);
91      locked_printf("* TA3: sent RTEMS_EVENT_6 to TA1.\n");
92      /*Send EVENT_7 to Task2 */
93      sc = rtems_event_send(
94          ctx->task[1],
95          RTEMS_EVENT_7);
96      rtems_test_assert(sc == RTEMS_SUCCESSFUL);
97      locked_printf("* TA3: sent RTEMS_EVENT_7 to TA2.\n");
```

Figure 52 Event sending

Task 3, before waking Task 2 which is waiting to receive EVENT_7, sends an event to Task 1, only with the flag RTEMS_EVENT_6. This is expected not to wake Task 1, as the latter is waiting on both event 6 AND 7. The first parameter on rtems_event_send() is the task ID to which the event is meant to be sent.

```
44      sc = rtems_event_receive(
45          RTEMS_EVENT_6 | RTEMS_EVENT_7,
46          RTEMS_WAIT | RTEMS_EVENT_ALL,
47          RTEMS_NO_TIMEOUT,
48          &event_out);
49      rtems_test_assert(sc == RTEMS_SUCCESSFUL);
```

Figure 53 Event receive

The image 54 is the directive Task 1 calls*, rtems_event_receive(*), in order to wait for the specified events. We can choose if the task waits or not for the events by specifying RTEMS_WAIT or RTEMS_NO_WAIT. We also decide if we want to wait for any one of the events or if both are required. We require both to proceed by using the RTEMS_EVENT_ALL flag. Last, the task will wait forever since we defined the macro RTEMS_NO_TIMEOUT on the parameter used to specify the time Task 1 will be waiting for the events.

```
64          sc = rtems_event_receive(
65              RTEMS_EVENT_7,
66              RTEMS_WAIT | RTEMS_EVENT_ALL,
67              RTEMS_NO_TIMEOUT,
68              &event_out);
69          rtems_test_assert(sc == RTEMS_SUCCESSFUL);
70
71          locked_printf("* TA2: finished.\n");
```

Figure 54 Event receive

Task 2 will have pretty much the same configuration, as it is visible on image 55. The only difference being the event this task is waiting for.

```
*** BEGIN OF TEST SMP_TEST_DEV 04 ***
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)

************ ********* ************
* CPU 3 running task Init.
id 167837698
id 167837699
id 167837700
* TA1: finished.
* TA3: sent RTEMS_EVENT_6 to TA1.
* TA2: finished.
* TA3: sent RTEMS_EVENT_7 to TA2.
* TA3: sent RTEMS_EVENT_6 and RTEMS_EVENT_7 to TA1.
* TA1: finished.
************ ********* ************


*** END OF TEST SMP_TEST_DEV 04 ***
```

Figure 55 Events sample - output

8 Mine Control Case Study

At the end, we wanted to develop a real-time system on a dual-core SMP configuration to evaluate our knowledge, test the behaviour and analyse the performance of RTEMS with a complex system, and as there are not many available case studies that explore the capabilities of SMPs, it was proposed to adapt the academic case study, Mine Control. It is concerned with the development of an embedded system software that controls and monitors a simplified pump system for a mining environment with several safety requirements.

8.1 Analysis

The first aspect was to understand the point and all the specific requirements of this case study. The system controls the water level in a sump, if the water reaches a certain level, a sensor will inform the system and if the safety requirements allow, it enables the pump motor to pump water out of the sump.

A device will also verify if the water is flowing, allowing to check if the pump is correctly working. The motor can be stopped in two situations, if the water drops to a certain limit, or when a failure exists within the safety requirements.

Since it is not safe to cut coal or operate the pump with certain levels of methane and carbon monoxide in the air, safety requirements must be guaranteed. They are achieved by the environment monitoring that is responsible for detecting the level of methane, carbon monoxide in the air and the air flow. the values are gathered from readings of external sensors.

In case of a reading from the methane level that exceeds a critical threshold, the pump must be disabled to avoid explosions, to this pump shutdown, a deadline must be respected. It is described by the relationship of the methane period T, the rate at which methane can accumulate R, the safety margin between the level of methane regarded as critical M and the level at which it explodes D, being expressed by the following inequality:

$$\mathrm{R}(T + D) < \mathrm{M}$$

So, until now, five external devices were identified, three of them to air monitoring table 10, and the other two to the water control table 11.

Table 10 Air Monitoring Devices

Device
Methane Sensor
Carbon Monoxide Sensor
Air-Flow Sensor

Device
High-Low Water Sensor
Water-Flow Sensor

Table 11 Water Control devices

The devices have a legislated period to cap the information from the outside world, so, all of them have a defined periodicity, worst case execution time (WCET), priority and a constrained deadline. As the methane was the most critical reading, it logically has the shortest period and deadline, while the High-Low Water has the longest period and deadline.

The priorities are assigned with a deadline-monotonic priority assignment policy, that is, the priorities are assigned inversely to the deadline, tasks with shortest deadline have a higher priority. The information is showed in table 12.

Device	Period	Deadline	Priority	WCET
Methane Sensor	80	35	10	12
Carbon Monoxide Sensor	100	60	12	10
Air-Flow Sensor	100	100	13	10
High-Low Water Sensor	6000	200	14	40
Water-Flow Sensor	1000	40	11	20

Table 12 Periodic tasks Information

8.1.1 Schedulability

As mentioned before, there are still several problems with the use of locking protocols with dynamics priority scheduling algorithms, so we had to discard the use of EDF to develop the case study. We chose Deterministic Priority scheduler, a fixed priority preemptive scheduler.

Before going forward with the development of the case study, we had to verify if the system was schedulable, that is, if all tasks were able to run without the failure of any deadline. So, for this test we used the famous sufficient, but overly pessimistic, RTA-Based schedulability test for Multiprocessor systems scheduled with fixed priority []. Being sufficient means if the test passes, the task set is schedulable. Otherwise it would be necessary to find another schedulability test less pessimistic.

$$\mathrm{R_k^{max}} \leftarrow \mathrm{C_k} + \frac{1}{\mathrm{m}} \sum_{\tau_{j \in hp(k)}} \left(\left\lceil \frac{R_k^{max}}{\mathrm{T}_j} \right\rceil \mathrm{C}_j + \mathrm{C}_j \right)$$

Whit this equation, a task set τ scheduled with a fixed priority. A bound on the maximum response time R_k^{max} of a task $\tau_k \in \tau$ is derived by the fixed point reached, by iteratively repeating the equation ,where $hp(k)$ is the set of tasks with priority higher then $\tau_k's$, with initial value of $R_k^{max} = \mathrm{C}_k$, being $\mathrm{C}_k = WCET_k$. The system is schedulable if the condition $R_k^{max} \leq D_k$ is met for every $\tau_k \in \tau$.

The *m* on the formula stands for the number of processors in the system (in this case, 2). On iteration *n+1,* for each task, the R_k^{max} value inside the brackets is replaced by the R_k^{max} calculated on iteration *n*. The C_j stands for the WCET of the tasks $\in hp(k)$.

According to this test, since we are working on a dual-core system the first two tasks with the higher priority (CH4 sensor, water-flow sensor) will not have their execution interfered by other tasks, so their maximum response time is there actual worst-case execution time.

$$R_1{}^{MAX} = 12$$

$$R_2{}^{MAX} = 10$$

On following iterations, when trying to calculate the response time of a task we will have in consideration all the higher priority tasks existing (since they can interfere with the execution of lower priority ones). We reach a result on each iteration (max. response time for the task) when the results from the two last iterations converge OR if the response time calculated exceeds the deadline of the task. In the last case the schedulability of our task set would not be possible to prove. (It would be needed another less pessimistic test). Following this analysis, we go on to CO sensor task:

$$R_3{}^1 = 10 + \frac{1}{2}\left[\left(\left\lceil\frac{10}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{10}{1000}\right\rceil * 10 + 10\right)\right] = 32$$

$$R_3{}^2 = 10 + \frac{1}{2}\left[\left(\left\lceil\frac{32}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{32}{1000}\right\rceil * 10 + 10\right)\right] = 32$$

On the second iteration we can see the result is the same as that on first iteration. And since this value is below the task deadline we go on to test the next task.

$$R_4{}^1 = 10 + \frac{1}{2}\left[\left(\left\lceil\frac{10}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{10}{1000}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{10}{100}\right\rceil * 10 + 10\right)\right] = 42$$

$$R_4{}^2 = 10 + \frac{1}{2}\left[\left(\left\lceil\frac{42}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{42}{1000}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{42}{100}\right\rceil * 10 + 10\right)\right] = 42$$

The above calculations made for the response time of Air-flow sensor task converge again, this time on value 42. Since 42 is less than 100 (deadline) it's still possible to schedule our task set until this point.

$$R_5{}^1 = 40 + \frac{1}{2}\left[\left(\left\lceil\frac{20}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{20}{1000}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{20}{100}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{20}{100}\right\rceil * 10 + 10\right)\right] = 82$$

$$R_5{}^2 = 40 + \frac{1}{2}\left[\left(\left\lceil\frac{82}{80}\right\rceil * 12 + 12\right) + \left(\left\lceil\frac{82}{1000}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{82}{100}\right\rceil * 10 + 10\right) + \left(\left\lceil\frac{82}{100}\right\rceil * 10 + 10\right)\right] = 82$$

On the last task to be evaluated, High-low water sensor, its visible that the values still converge, and the calculated result also lies below the constrained deadline (200) so, we can now guarantee the schedulability of our task set, with the properties defined above, on a dual-core environment.

8.2 Design

The design process of a real-time application is a complex task, that is further complicated by the spreading of this activity to a set of processors, instead of just one. When making the adaptation decisions for designing this use case we had in attention the use of standard RTEMS software components, as this significantly reduces the time required to develop real-time applications.

In this case the whole system of sensors and actuators is meant to be implemented on a single RTEMS application. This is possible by creating RTEMS tasks that simulate the readings of the sensors. The team was faced with several adaptation problems due to our execution environment, this includes the programming language used, (Because of the limitations C impose i.e., lack of interfaces, the operating system and the hardware.

On top of that, the RTEMS real-time multitasking executive allows an application to be cast into a set of logical, autonomous processes/tasks which become easily manageable, on the design and implementation steps. For these reasons, the HRT-HOOD design method does not fit quite right. Yet, the approach of our design method is fundamentally based on it.

Initially this adaptation was proposed to be implemented on an actual physical, dual-core board, this plan however never came to be. So, we were left with using the, now familiar, QEMU to simulate our board (note: the decision was to keep two cores).

After the requirements gathered during the analysis phase, we already knew the existence of five RTEMS tasks, that represented the simulation of external devices. So, the next step was to think on all the other tasks, and how we were going to represent the pump and methane monitoring and the relationship between them.

We came up with the following interaction between all the entities of our system, figure 57, the pump and the CH4 status on the environment represent the critical objects of our system, requiring a careful synchronization when accessing it.

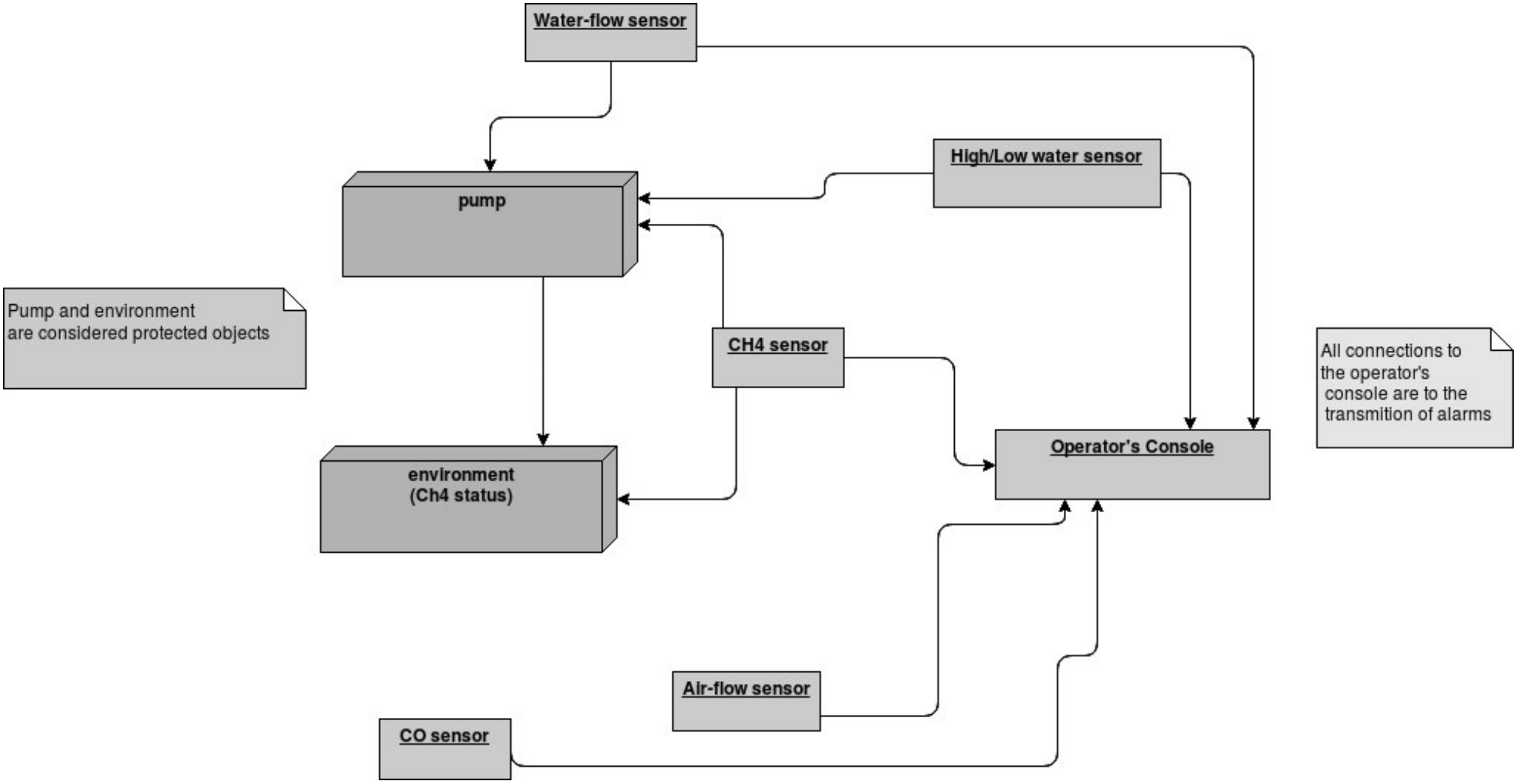


Figure 56 Interaction between the different entities of the case study

All the tasks referred as sensors send information to the operator console when dangerous values are simulated. This task will later print that information into the terminal.

The decision was to make the high-low water sensor also responsible for simulating the water level readings, as creating dedicated periodic “simulation” task for the water would change our schedulability analysis (and making our application unschedulable on the targeted machine).

The tasks “CH4 sensor” and “High-low water sensor” will have the responsibility of turning the pump on or off, accordingly. The first idea was to use the RTEMS software signals (ASR) and handlers to manage the pump. This solution however proved to be non-feasible for our real-time constraints, as RTEMS signals are implemented in a way that the task will only handle the signal sent when entering a processor, which would most likely cause the failure of the “shutdown pump” deadline.

So, the only solution found without losing any real-time capabilities and overloading the hardware was the use of structures that held the pump and methane state.The High-low water task will make changes to the pump state: ON or OFF, if the safety requirements are met.

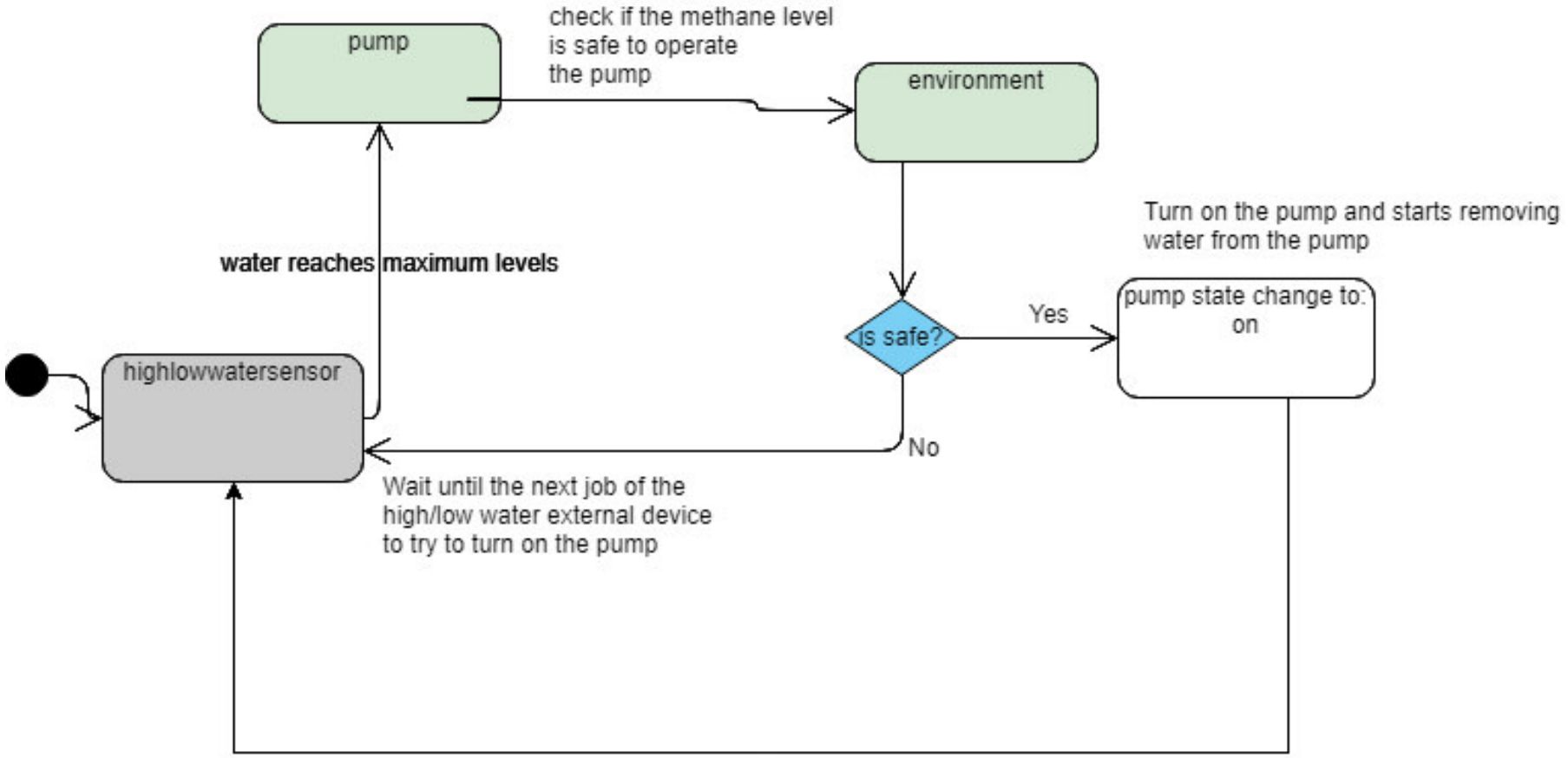


Figure 57 State diagram – high-low water task turning the pump ON

The check for safety is made to our protected object, CH4status. To turn the pump off, the task only needs to check the pump state on the protected object (as turning the pump off represents no danger, this action does not require previous validations).

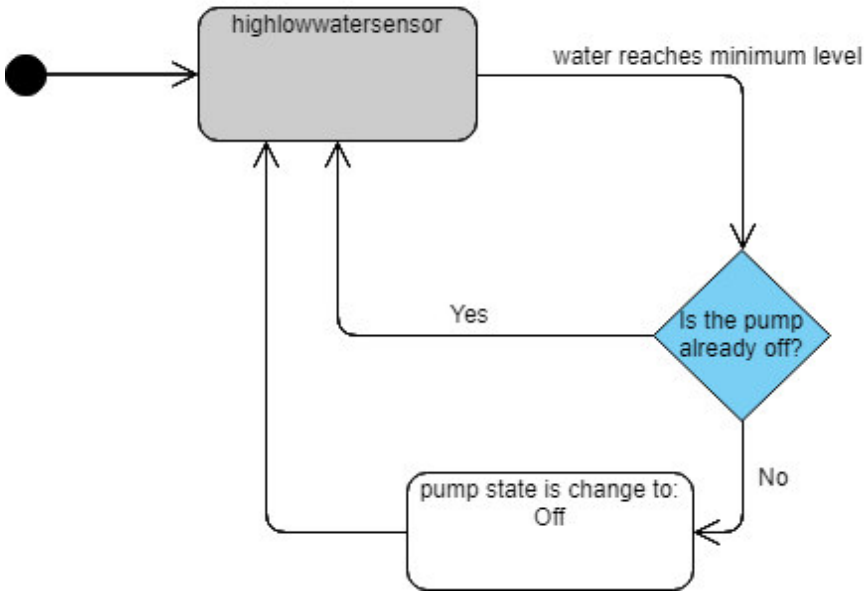


Figure 58 State diagram – high-low water task turning the pump OFF

The changes CH4 sensor task makes to the objects state are a little more complex than the previous presented.

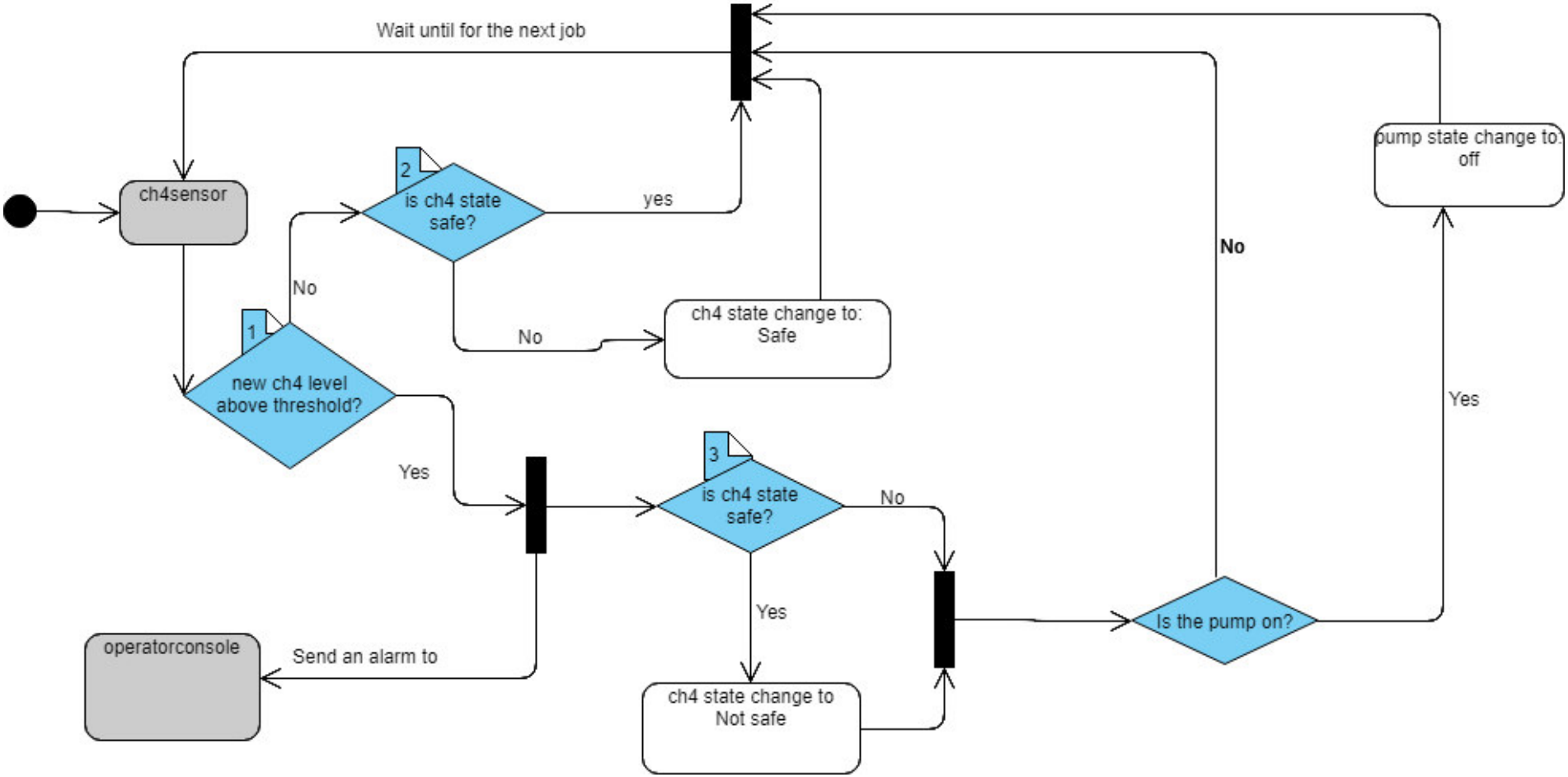


Figure 59 State diagram - CH4 sensor changing pump and ch4 status

This sensor will manage the CH4 state: SAFE or NOT SAFE and can also turn the pump OFF. If the pump is working while a new ch4 level reading above the critical threshold enters de system this task will change the pump state to OFF and the CH4 state to NOT SAFE. For performance reasons, the ch4 state is checked first at points 2 and 3 on the image, before making the changes.

8.3 Deployment

When starting the implementation, we first developed the system configuration. Selecting the scheduler, Deterministic Priority SMP, to then allocate it to the available processors, we also had to configure the floating-point unit and extra stack size needed, as it can be seen in figure 60

```
#define CONFIGURE_SCHEDULER_PRIORITY_SMP

#include <rtems/scheduler.h>

RTEMS_SCHEDULER_PRIORITY_SMP(a, CONFIGURE_MAXIMUM_PRIORITY);

#define CONFIGURE_SCHEDULER_TABLE_ENTRIES \
  RTEMS_SCHEDULER_TABLE_PRIORITY_SMP(a, rtems_build_name('P', 'R', 'I', ' '))

#define CONFIGURE_SCHEDULER_ASSIGNMENTS \
  RTEMS_SCHEDULER_ASSIGN(0, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY),\
  RTEMS_SCHEDULER_ASSIGN(0, RTEMS_SCHEDULER_ASSIGN_PROCESSOR_MANDATORY)



#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_INIT_TASK_ATTRIBUTES RTEMS_FLOATING_POINT
#define CONFIGURE_INIT_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 10)
#define CONFIGURE_EXTRA_TASK_STACKS (3*RTEMS_MINIMUM_STACK_SIZE)
#define CONFIGURE_INITIAL_EXTENSIONS RTEMS_TEST_INITIAL_EXTENSION

#ifdef SINGLE_PRECISION
#define FLOAT float

#else /* !SINGLE_PRECISION */
#define FLOAT double

#endif /* SINGLE_PRECISION */


#include <rtems/confdefs.h>
```

Figure 60 System configuration

We continued our case study, whit the implementation of the header files and structures that would represent the protected objects, pump and environment (ch4status). Those objects were mainly used to store several states and the tick period on which occurred the last state change.

```
typedef struct{
    bool state;
    bool state_before;
    rtems_interval last_state_change;
} Pump;

typedef struct{
  FLOAT ch4_value;
  bool state;
} CH4;
```

Figure 61 Protected objects

As in the others RTEMS application, the init task was configured to be the first one to be executed in our system, she had the responsibility to create other tasks and to setup the environment. The fundamental parts of the environment setup were:

- Creation of semaphores with MrsP protocol -> those semaphores are crucial to synchronize the access to the protected objects, one to access the pump and other to control the ch4status.

```
sc = rtems_semaphore_create(
  rtems_build_name('S', 'P', 'U', 'M'),
  1,
  RTEMS_MULTIPROCESSOR_RESOURCE_SHARING | RTEMS_BINARY_SEMAPHORE,
  PRIORITY_CEILLING,
  &pointer_id->sem_id[PUMP_SYNCHRO]
);
```

Figure 62 MrsP semaphore

- Data structures initialization -> initialization of the protected objects and other structures who are responsible to store other readings.

```
void (*struct_init_ptr)(void)=&Struct_initialization;
(*struct_init_ptr)();
```

Figure 63 Funtion pointer to initialize data structures

- Creation of a barrier -> to synchronize the periodicity creation for the simulated external devices tasks.
- Creation of a message queue -> used to send alarms when required.

When creating the tasks that would represent the external devices, we had to be careful with the task's attributes, since some had to use the floating-point unit to have decimal values.

```
/* create ch4 sensor */
create_task(index, CH4_SENSOR_PRIOITY,RTEMS_FLOATING_POINT);
sc=rtems_task_start(pointer_id->tasks_id[index], Ch4_sensor, (rtems_task_argument) index);
directive_failed(sc,"Error starting 1 \n");
```

Figure 64 Task Creation

```
rtems_task create_task(uint32_t index,rtems_task_priority priority,rtems_attribute atributes)
{
char val=' '+ index;
rtems_status_code sc;
  sc=rtems_task_create(
      rtems_build_name('T','A',val,' '),
      priority,
      (RTEMS_MINIMUM_STACK_SIZE),
      RTEMS_DEFAULT_MODES,
      atributes,
      &pointer_id->tasks_id[index]
    );
```

Figure 65 Task creation

When we started to implement the periodic tasks with the help of the rate monotonic manager, we were confronted with a deadline issue since RTEMS does not offers the possibility to apply constrained deadline, it only accepts implicit deadlines (relative deadline = period). So, we had to find a way to turn around this problem. A solution found was the use of timers, at the beginning of the periodic code, a timer was set to fire when reached the tick moment correspondent to the absolute deadline. if reached, a specific routine was invoked, and sent an alarm through the message queue informing the deadline failure, figure 67. If the periodic code ends without reaching the deadline, the timer is cancelled to be reinitialize in the next job, figure 66.

```
rtems_rate_monotonic_period(period_id, periodicity);
rtems_timer_fire_after(timer,deadline,time_expired_ch4_sensor,NULL);
Ch4_period_code(buffer);
rtems_timer_cancel(timer);
```

Figure 66 Timer

```
rtems_timer_service_routine time_expired_ch4_sensor()
{
  rtems_status_code sc;
  sprintf(deadlinebuffer,"1- METHANE SENSOR DEADLINE MISSED, MOMENT %i ",rtems_clock_get_ticks_since_boot());
  sc=rtems_message_queue_urgent(message_queue_id,(void *)&deadlinebuffer,strlen(deadlinebuffer));
```

Figure 67 Timer service routine

All the tasks communicated with the operator, sending alarms of dangerous readings and deadline failures through a message queue. We used a message queue since it

already offers synchronization and in case of deadline failure we could send the alarm as urgent, it would go directly to the front of the queue to then be handled.

On the next chapters all tasks will be further explain.

8.3.1 CH4

The methane task starts by simulating a new ch4 value, according with the probability, followed by a call to the environment where it alters the ch4status.

```
if(probability)
{
  if(ch4status->ch4_value<=HIGH_LEVEL_POSSIBLE-CH4_RATE)
  {
     new_value_add=ch4status->ch4_value+CH4_RATE;
    (*change_value_ch4_ptr)(new_value_add);

  }
}
else
{
  if(ch4status->ch4_value>=LOW_LEVEL_POSSIBLE+CH4_RATE)
  {
      new_value=ch4status->ch4_value-CH4_RATE;
    (*change_value_ch4_ptr)(new_value);

  }
}
```

Figure 68 Methane simulation

After simulating, the ch4 task verifies if the new value was safe or not, a value between 5% and 15% of methane in the air was considered not safe. If the new reading was not safe, it informs the operator through the function *alarm* that uses a message queue, in this case it also has the responsibility to shut down the pump to avoid an explosion going directly to the pump through the function *turn_pump_off()*. The sequence described can be seen in figure 69.

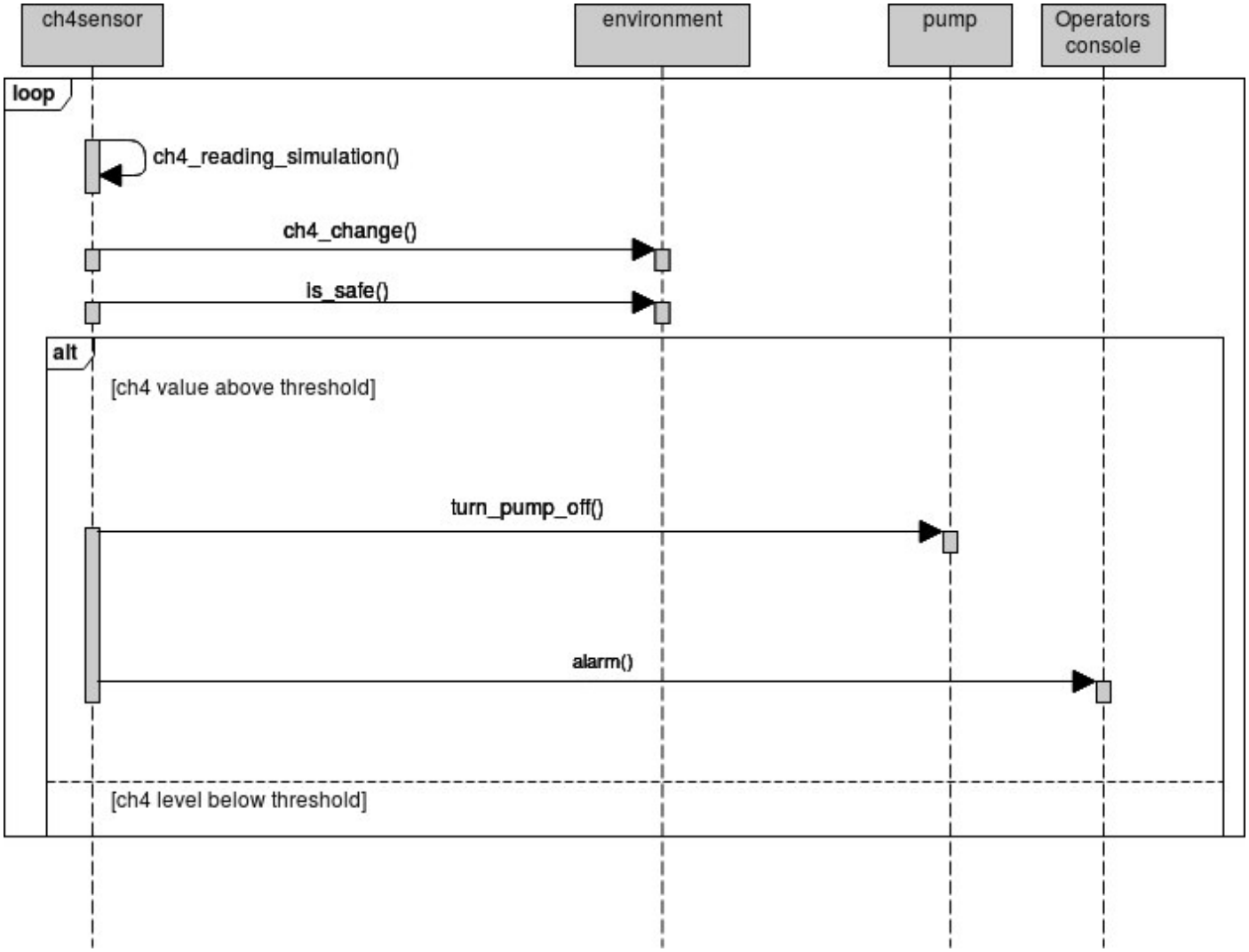


Figure 69 CH4 Sequence diagram

8.3.2 CO and Air flow

The carbon monoxide task was easier to develop compared with the methane task, since this one does not have any relationship with the pump, it only simulates the new co value, storing it the environment, and in case of dangerous reading it sends an alarm to the operator.

Meanwhile, the job of the air flow was extremely similar to the monoxide carbon task, it only simulates the new value and if it was considered dangerous it sends it to the operator, this sequence can be verified in figure 71.

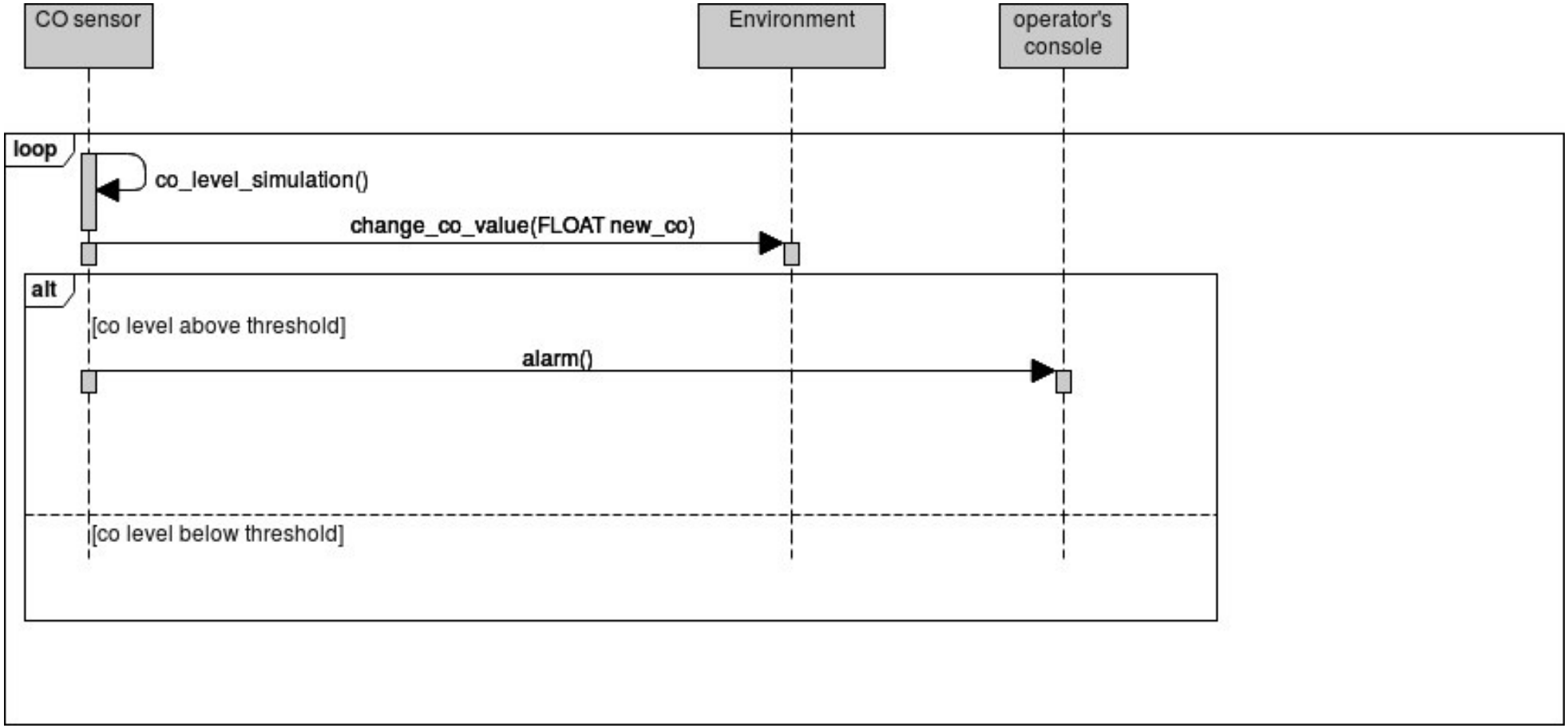


Figure 70 CO sequence diagram

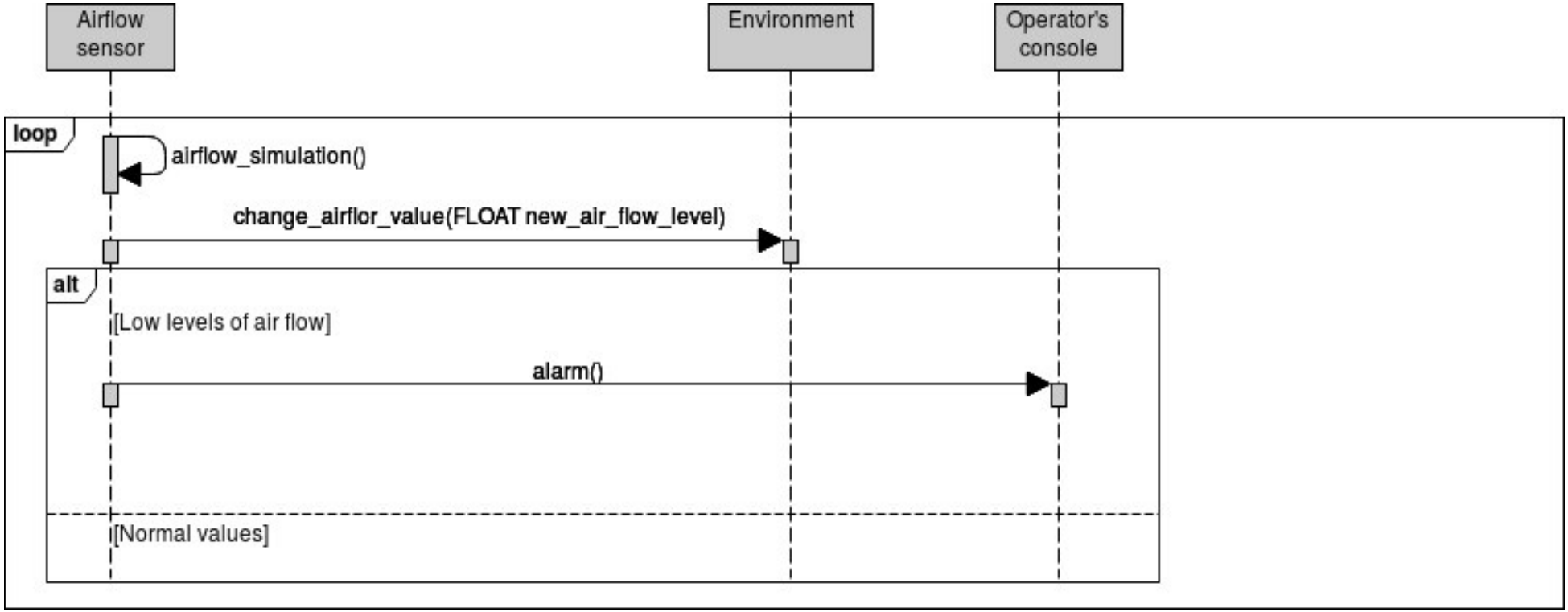


Figure 71 Air flow sequence diagram

8.3.3 Water Flow

This task was responsible to verify if there occurred any unexpected behaviour with the water flow, we done it by simulating a value within a probability, and if the pump was on, it means that the water was not flowing but should be, and the opposite if the pump was off. This sequence is represented in figure

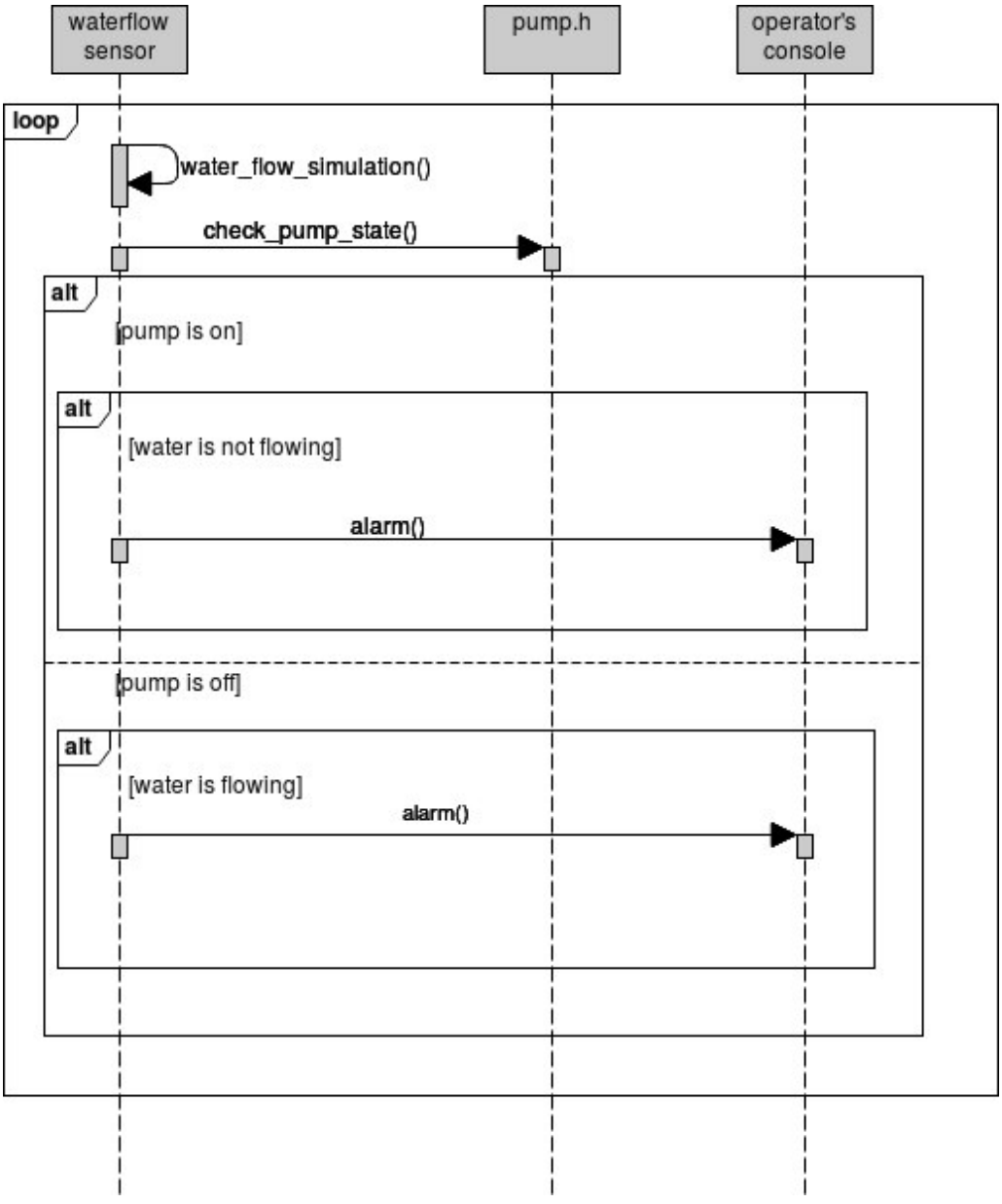


Figure 72 Water flow sequence diagram

8.3.4 HighLow Water

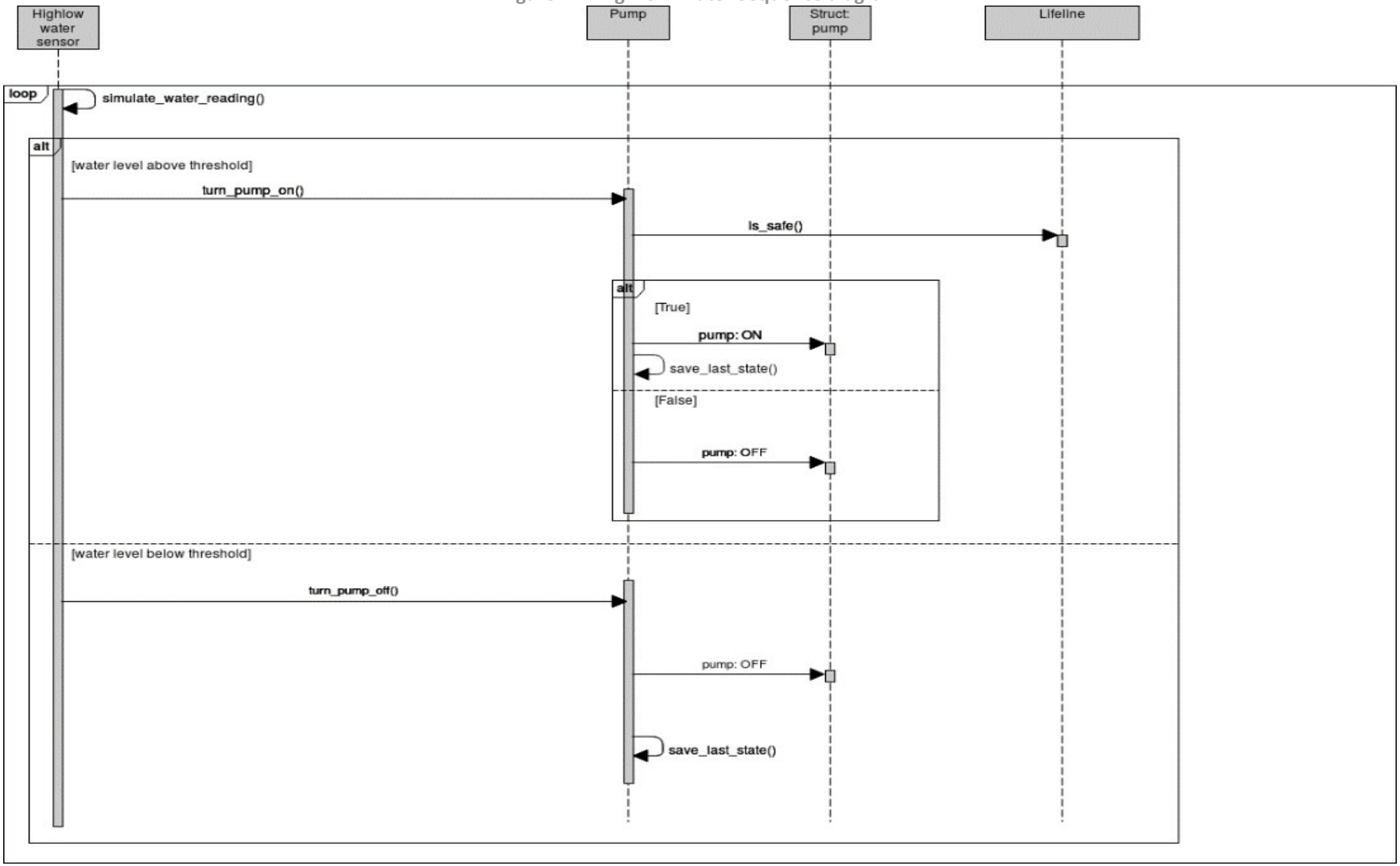
This external device had the burden to simulate the water and then verify if its level was above or under a certain level. To simulate the water, as it can be seen in figure 73, we first started by checking if we had a uniform period, that is, if there was a change in state of the pump in the last period. If no change occurred, we could simply increment or reduce the water level according with the defined rates. If a changed happened, we had to calculate the amount of water retrieved when the pump was on and the amount gained when the pump state was off.

Figure 73 Water simulation

```
static void water_simulation()
{

  rtems_status_code sc;
  uint32_t amount_removed, amount_filled;

  if(period_uniform())
  {
    if(pump_is_on())
    {

      if((water->water_lvl - (HIGHLOWWATERPERIODICITY  * EXT_RATE))< WATER_ERROR_LEVEL)
      {

        water->water_lvl = WATER_ERROR_LEVEL;
      }
      else
      {
        water->water_lvl = water->water_lvl - (HIGHLOWWATERPERIODICITY * EXT_RATE);}
      }
    else
    {
      water->water_lvl = water->water_lvl + (HIGHLOWWATERPERIODICITY * FILL_RATE);
    }
  }
  else
  {



    sc = rtems_semaphore_obtain(pointer_id->sem_id[PUMP_SYNCHRO], RTEMS_WAIT, RTEMS_NO_TIMEOUT);
    directive_failed(sc,"Error sem \n");

    amount_removed = (p_pump->last_state_change - (  rtems_clock_get_ticks_since_boot() -HIGHLOWWATERPERIODICITY)) * EXT_RATE;
    amount_filled = (rtems_clock_get_ticks_since_boot() - p_pump->last_state_change) * FILL_RATE;

    sc = rtems_semaphore_release(pointer_id->sem_id[PUMP_SYNCHRO]);

    directive_failed(sc,"Error sem \n");
    water->water_lvl = water->water_lvl + amount_filled - amount_removed;

  }

}
```

After simulating a new value, this task had to check if the water reached certain levels. If the level was above a maximum value, it had to verify through the function *is_safe()* if the methane was not considered dangerous, if not, the pump could be turned on. And if the level was under the minimum value, the highlowwater task turned the pump off with the function *turn_pump_off().* The sequence diagram in figure 74 describes the connection of this task with the pump and the environment.

Figure 74 HighLow Water Sequence diagram



8.4 Results

For this section, we initially started by only printing the preemptions to verify if no period failures occurred, that is, if all the periodic tasks entered the processor when they should according with the defined periodicity. But first, we configured the circular buffer to only store the preemptions related to the task we indicated, this eased our job of verifying if the tasks had the correct periodicity behaviour.

In the figure 75, we can verify that there was no period failure with the methane task, known by the system as thread 167837698, it enters a CPU each 80 ticks. In figures 76 and 77 is presented the preemptions of the threads 1678376989 and 167837700, they represents the monoxide carbon and air flow tasks respectively, both do not present any period failure as they enter the processor each 100 ticks.

```
Index 1 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 12
Index 2 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 89
Index 3 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 89
Index 4 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 169
Index 5 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 170
Index 6 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 249
Index 7 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 250
Index 8 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 329
Index 9 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 330
Index 10 , leaving thread  167837700 , entering thread 167837698  cpu 1 time 409
Index 11 , leaving thread  167837698 , entering thread 167837700  cpu 1 time 409
Index 12 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 489
Index 13 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 490
Index 14 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 569
Index 15 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 570
Index 16 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 649
Index 17 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 650
Index 18 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 729
Index 19 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 730
Index 20 , leaving thread  167837700 , entering thread 167837698  cpu 1 time 809
Index 21 , leaving thread  167837698 , entering thread 167837700  cpu 1 time 810
Index 22 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 889
Index 23 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 890
Index 24 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 969
Index 25 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 969
Index 26 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 1049
Index 27 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 1050
Index 28 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 1129
Index 29 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 1130
Index 30 , leaving thread  167837700 , entering thread 167837698  cpu 1 time 1209
Index 31 , leaving thread  167837698 , entering thread 151060481  cpu 1 time 1210
Index 32 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 1289
Index 33 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 1290
Index 34 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 1369
Index 35 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 1370
Index 36 , leaving thread  151060482 , entering thread 167837698  cpu 0 time 1449
Index 37 , leaving thread  167837698 , entering thread 151060482  cpu 0 time 1450
```

Figure 75 CH4 buffer result

```
Index 10 , leaving thread  167837701 , entering thread 167837699  cpu 1 time 19
Index 11 , leaving thread  167837699 , entering thread 167837700  cpu 1 time 19
Index 12 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 119
Index 13 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 120
Index 14 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 219
Index 15 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 220
Index 16 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 319
Index 17 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 320
Index 18 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 419
Index 19 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 419
Index 20 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 519
Index 21 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 520
Index 22 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 619
Index 23 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 620
Index 24 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 719
Index 25 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 720
Index 26 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 819
Index 27 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 819
Index 28 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 919
Index 29 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 919
Index 30 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1019
Index 31 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1019
Index 32 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1119
Index 33 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1119
Index 34 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1219
Index 35 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1219
Index 36 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1319
Index 37 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1319
Index 38 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1419
Index 39 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1419
Index 40 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1519
Index 41 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1519
Index 42 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1619
Index 43 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1619
Index 44 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1719
Index 45 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1719
Index 46 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1819
Index 47 , leaving thread  167837699 , entering thread 151060482  cpu 0 time 1819
Index 48 , leaving thread  151060482 , entering thread 167837699  cpu 0 time 1919
```

Figure 76 CO buffer result

```
Index 2 , leaving thread  167837699 , entering thread 167837700  cpu 1 time 14
Index 3 , leaving thread  167837700 , entering thread 167837702  cpu 1 time 15
Index 4 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 114
Index 5 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 114
Index 6 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 214
Index 7 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 215
Index 8 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 314
Index 9 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 315
Index 10 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 414
Index 11 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 415
Index 12 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 514
Index 13 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 515
Index 14 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 614
Index 15 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 615
Index 16 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 714
Index 17 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 715
Index 18 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 814
Index 19 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 814
Index 20 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 914
Index 21 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 915
Index 22 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1014
Index 23 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1014
Index 24 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1114
Index 25 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1116
Index 26 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1214
Index 27 , leaving thread  167837700 , entering thread 151060482  cpu 0 time 1214
Index 28 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1314
Index 29 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1314
Index 30 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1414
Index 31 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1414
Index 32 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1514
Index 33 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1515
Index 34 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1614
Index 35 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1614
Index 36 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1714
Index 37 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1715
Index 38 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1814
Index 39 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1815
Index 40 , leaving thread  151060481 , entering thread 167837700  cpu 1 time 1914
Index 41 , leaving thread  167837700 , entering thread 151060481  cpu 1 time 1914
```

Figure 77 Air flow buffer result

And for last, the water flow, 167837701, and the highlow water, 167837702, behaved as expected, since both of them do not presented any periodic failure, the water flow entered a processor every 1000 ticks, figure 78, and the highlow water task entered each 6000 ticks, figure 79.

```
Index 2 , leaving thread  151060481 , entering thread 167837701  cpu 1 time 10
Index 3 , leaving thread  167837701 , entering thread 167837699  cpu 1 time 12
Index 4 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 1011
Index 5 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 1011
Index 6 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 2011
Index 7 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 2011
Index 8 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 3011
Index 9 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 3011
Index 10 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 4011
Index 11 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 4013
Index 12 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 5011
Index 13 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 5011
Index 14 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 6011
Index 15 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 6012
Index 16 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 7011
Index 17 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 7013
Index 18 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 8011
Index 19 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 8011
Index 20 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 9011
Index 21 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 9011
Index 22 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 10011
Index 23 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 10011
Index 24 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 11011
Index 25 , leaving thread  167837701 , entering thread 151060482  cpu 0 time 11012
Index 26 , leaving thread  151060482 , entering thread 167837701  cpu 0 time 12011
```

Figure 78 Water buffer result

```
Index 2 , leaving thread  167837700 , entering thread 167837702  cpu 1 time 11
Index 3 , leaving thread  167837702 , entering thread 151060481  cpu 1 time 14
Index 4 , leaving thread  167837700 , entering thread 167837702  cpu 1 time 6011
Index 5 , leaving thread  167837702 , entering thread 151060481  cpu 1 time 6013
Index 6 , leaving thread  151060482 , entering thread 167837702  cpu 0 time 12011
Index 7 , leaving thread  167837702 , entering thread 167837700  cpu 0 time 12011
Index 8 , leaving thread  167837700 , entering thread 167837702  cpu 0 time 12011
Index 9 , leaving thread  167837702 , entering thread 151060481  cpu 0 time 12011
Index 10 , leaving thread  151060482 , entering thread 167837702  cpu 0 time 18011
Index 11 , leaving thread  167837702 , entering thread 167837700  cpu 0 time 18011
Index 12 , leaving thread  167837699 , entering thread 167837702  cpu 1 time 18011
Index 13 , leaving thread  167837702 , entering thread 151060481  cpu 1 time 18012
Index 14 , leaving thread  151060482 , entering thread 167837702  cpu 0 time 24011
Index 15 , leaving thread  167837702 , entering thread 167837700  cpu 0 time 24011
Index 16 , leaving thread  167837699 , entering thread 167837702  cpu 1 time 24013
Index 17 , leaving thread  167837702 , entering thread 167837698  cpu 1 time 24014
```

Figure 79 HighLow water buffer result

Despite the scheduling test affirmed that no deadline failure was expected to occur, we decided anyway to test if any deadline failure could occur within our system, so, we removed the output related to the dangerous readings to isolate the deadline alarms. And as it can be seen in figure 80, no alarm related to the deadline failure was received by the operator.

```
*** BEGIN OF TEST minecontrol ***
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)


*** END OF TEST minecontrol ***
```

Figure 80 Mine Control Deadline alarms

To also test the behaviour of the system, we decided to introduce a deadline failure in one task and verify what could be the sequels. So, for that, we provoked an error in the water flow task. In figure 81 are presented the results, and we can see the alarms sent to the operator, as we introduced a short failure it had no implication on the execution of the other tasks.

```
*** BEGIN OF TEST minecontrol ***
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)

1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 15
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 1015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 2015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 3015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 4015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 5015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 6015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 7015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 8015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 9015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 10015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 11015
1- DEADLINE WATER FLOW SENSOR DEADLINE MISSED, TIME 12015


*** END OF TEST minecontrol ***
```

Figure 81 Water Flor Deadline Alarm

On the appendixes section 11, it's possible to check the output that result from the execution of our application. We looked to increase the number of alarms raised due to the levels of methane, as this is the more critical value to have in consideration. It is visible that no alarm is raised due to the missing of deadlines.

There is an important distinction to be made between the "moment" and the "actual time" columns. The moment refers to the time at which the alarm was sent. The actual time refers to the time in which the operator received the alarm. Note that the maximum deviation that ever occurs between the two values is in the magnitude of 1 tick, which means there is almost no delay between the raise of the alarm and the reach of this to the console.

If this system were to communicate via a network interface to the operator's terminal, this would most likely have a much larger delay.

9. Conclusions

The current day situation regarding symmetric multiprocessor environments is starting to look promising. Only now, solutions that allow us to efficiently, and correctly harness the larger processing power of multiprocessors are arriving to the embedded systems world.

This project aims of exploring the area of study by allowing us to test the RTEMS mechanisms in SMP proved to be a real challenge for a team that was just finishing the current computer engineering degree. The test-suites idea served to make sure the directives offered by RTEMS worked as intended. Since the SMP platform already brings enough challenge as it currently is, it is of extreme importance to make sure the functions of the RTOS do not show unpredictable behaviours.

The circular buffer deployment doesn't bring much overhead to the system, even though it is there, it's quite negligible. In a simple manner, the project can be split in 3 main objectives that are tightly related: The test-suites, the circular buffer and the use case development. The test-suites and buffer development really helped us get an understanding on what a RTOS should support, and what are its main concerns. Mainly because this required quite a long investigation though the RTEMS docs. The final step that contemplated the mine drainage case study, put us in contact with the schedulability analysis of a system and the much more task and time constrained environments this kind of programming takes.

9.1 Accomplished Objectives

In the end, the main objectives touched on section 2 were accomplished.

First, the test-suites original proposed. All the schedulers implemented on RTEMs were targeted during the development of these samples. The synchronization mechanisms that were missing samples (Events, Message Queues) were tested, as well as the MrsP (helping protocol), OMIP and the barrier directives. The value of the samples developed lies on the possible use they can bring to support future RTEMS applications. The header files of each test-suite can be consulted when dealing with the same or similar configurations.

The work related with the buffer deployment on the RTEMS kernel incentivised our research on the innerworkings of the RTOS and brought a better understanding of its behaviour. The changes we came up with allows to get access to the thread preemptions happening on the processors at runtime, specifically where and when they happen. This feature is only available for SMP configurations.

The development of the mine drainage use case represented an important objective achieved by the group. The solution implemented resulted on a small RTEMS application for SMP systems with real-time capabilities.

In the end, the application is capable of simulating readings from water and air sensors and then managing the state of the pump motor, turning it on or off. This management to the motor is made according to the water level and CH4 read. Despite the late arrival of SMP systems to the real-time world, this work looked to contribute to this field by researching the topic and going for a new adaption of an old case study. The schedulability analysis allow us to prove the system is schedulable, and this can be attested empirically on subsection 8.4 where it is visible that no task misses its imposed deadline.

9.2 Limitations and Future Work

One interesting extension of this work related with the use case that immediately pops out is using real sensors communicating with the application to obtain the readings. This would result on a much more cohesive application and would leave us with more schedulability options to try out different designs.

Another interesting suggestion would be testing the performance of our solution so than, it would be possible to compare it against the original [22]. This would mean remove the whole “sensor simulation” too beforehand of course, otherwise the results would be meaningless. It would be possible to gather some interesting information about the operating system.

Regarding the sample tests, we would like to see them uploaded along with the rest of the samples present on the RTEMS official github page. From there, they would reach a much larger number of people and who knows could serve some purpose to anyone interested.

Even though the overall positive balance made, every academic work is bounded by some limitations. The fact that we had no chance to implement and test the case study on a physical board was a missed opportunity.

The limitations on our knowledge of embedded systems proved to make the writing of this report quite a challenging one. But one of the main purposes of this project was also to familiarize us with this field of study, and that was an accomplished goal. The approach taken to solve several design issues regarding the case study we think reflect this exactly and we end feeling like the main goals were attained.

10. Bibliography

[1] Real-Time Systems. (2018, May). Retrieved from https://www.cse.unsw.edu.au/~cs9242/08/lectures/09-realtimex2.pdf

[2] Difference Between RTOS and OS. (2018, June). Retrieved from http://www.differencebetween.net/technology/difference-between-rtos-and-os/

[3] Multiprocessors. (2018. May). Retrieved from https://www.cs.vu.nl/~ast/books/mos2/sample-8.pdf

[4] IEEE Xplore. (2018, June 3). Retrieved from https://ieeexplore.ieee.org/document/5347560/

[5] CISTER. (2018. May). Retrieved from http://www.cister.isep.ipp.pt/

[6] Burns, A; Wellings, March 2009, A. Real-Time Systems and Programming Languages (Fourth Edition), (pag. 547). Addison Wesley.

[7] RTOS comparison (2018. May). Retrieved from https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

[8] RTEMS Real Time Operating System (RTOS). (June). Retrieved from: https://www.rtems.org/

[9] RTEMS Classic API Guide 5.0.0 (2018, September). Retrieved from https://docs.rtems.org/branches/master/c-user/index.html

[10] Challenges in programming multiprocessor platforms. (June). Retrieved from http://www.mpsoc-forum.org/previous/2004/slides/goodacre.pdf

[11] Multiprocessing with real-time operating systems, (2018, June). Retrieved from: https://www.embedded.com/design/prototyping-and-development/4024574/Multiprocessing-with-real-time-operating-systems

[12] C programming language (2018. June). Retrieved from: http://www.dipmat.univpm.it/~demeio/public/the_c_programming_language_2.pdf

[13] Qemu (2018. June). Retrieved from: https://www.qemu.org/

[14] Sourcetrail (2018. June). Retrieved from: https://www.sourcetrail.com/

[15] Gnome commander (2018. June). Retrieved from: https://gcmd.github.io/

[16] RTEMS source builder (2018. June). Retrieved from: https://devel.rtems.org/wiki/Developer/Tools/RSB

[17] A Schedulability Compatible Multiprocessor Resource Sharing Protocol – MrsP. (September 2018). Retrieved from https://www-users.cs.york.ac.uk/burns/MRSPpaper.pdf

[18] Development Environment for Future LEON Multi-Core (Semptember 2018). Retrieved from http://microelectronics.esa.int/gr740/RTEMS-SMP-FinalReport-SpaceBelEmbeddedBrainsUnivPadova.pdf

[19] Reliable Software Technologies – ADA Europe 2015 (pag. 190). Retieved from: https://goo.gl/ZDwvw7

[20] A Fully Preemptible Multiprocessor Semaphore Protocol for Latency-Sensitive Real-Time Applications (September 2018). Retrieved from https://ieeexplore.ieee.org/document/6602109/

[21] Multicore Processing in the Avionics Industry. (2018, June 3). Retrieved from http://rtsl-edge.cs.illinois.edu/CMAAS17/media/talk_5.pdf

[22] Reliable Multicore Processors for NASA Space Missions. (2018, June 3).Retrieved from https://trs.jpl.nasa.gov/bitstream/handle/2014/41793/11-0119F.pdf?sequence=3

[23] The Use of Multicore Processors in Airborne Systems (MULCORS). (2018, June 3).Retrieved from https://www.easa.europa.eu/document-library/research-projects/easa20116

[24] Assurance of Multicore Processors in Airborne Systems (2018, June 3).Retrieved from: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/TC-16-51.pdf

[25] T-Kernel, Multi-core edition. (2018, July). Retrieved from: https://www.esol.com/embedded/download/pdf/datasheet_et-kernel_mce_mpo.pdf

[26] https://ptolemy.berkeley.edu/publications/cps.htm

[27] A Structured Design Method for Hard Real-time Systems (2018, August 10). Retrieved from http://beru.univ-brest.fr/~singhoff/ENS/UE_SEE/TP-AADL/PRJ-2017/hrt-hood.pdf

[28] Symmetric Multiprocessing (RTEMS). (2018, September). Retrieved from: https://docs.rtems.org/branches/master/c-user/symmetric_multiprocessing_services.html

[29] ASU Library | Digital Repository. (2018, June). Retrieved from: https://repository.asu.edu/attachments/137367/content/Bulusu_asu_0010N_14279.pdf

[30] Difference between Multiprogramming, Multitasking, Multithreading and Multiprocessing (June 2018) .Retrieved from https://www.8bitavenue.com/2012/10/difference-between-multiprogramming-multitasking-multithreading-and-multiprocessing/

[31] New schedulability analysis for MrsP (Agust 2018). Retrieved from http://eprints.whiterose.ac.uk/131970/1/RTCSA_2017_paper_6.pdf

[32] A New Configurable and Parallel Embedded Real-time Micro-Kernel for Multi-core platforms (June 2018) .Retrieved from https://people.mpi-sws.org/~bbb/events/ospert15/pdf/ospert15-p25.pdf

[33] What Is Priority Inversion (And How Do You Control It)? (June 2018). Retrieved from http://www.drdobbs.com/jvm/what-is-priority-inversion-and-how-do-yo/230600008

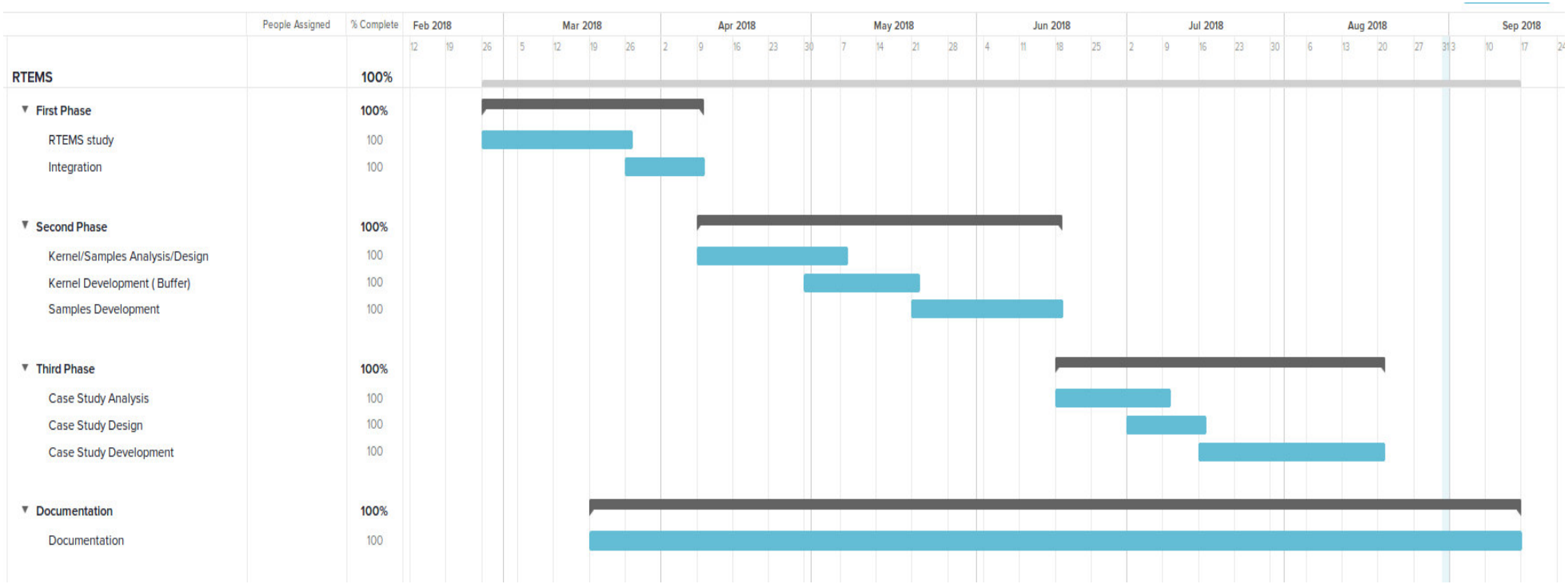
[34] Real-time programming with RTEMS (August 2018). Retrieved from http://beru.univ-brest.fr/~singhoff/ENS/USTH/TP/tp.html#Ref1

[35] How Hard is Partitioning for the Sporadic Task Model? (September 2018). Retrieved from https://ieeexplore.ieee.org/document/5366870/

[36] What really happened to the software on the Mars Pathfinder spacecraft? (August 2018). Retrieved from https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-spacecraft

11. Appendixes

11.1 Gantt Diagram



11.2 Mine Control Output

```
*** TEST VERSION: 5.0.0.b8c59353552c2504c0e71e1f6f81dfa4b2a96e37-modified
*** TEST STATE: EXPECTED-PASS
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API RTEMS_SMP
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB d255e812abd5d46d7cbbd4ef924f9d1267100c0b, Newlib 3.0.0)
id 167837698


 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.50, moment 4414 , actual time 4422
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.50, moment 6413 , actual time 6413
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.40, moment 6513 , actual time 6513
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.30, moment 6613 , actual time 6613
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.40, moment 6713 , actual time 6713
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.30, moment 6814 , actual time 6814
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.40, moment 6913 , actual time 6914
 2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.50, moment 7013 , actual time 7013
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 8015  , actual time 8015
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 9775  , actual time 9775
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 9855  , actual time 9855
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 9935  , actual time 9935
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 10015  , actual time 10015
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 10095  , actual time 10095
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 10175  , actual time 10175
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 10256  , actual time 10257
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 10415  , actual time 10415
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 10495  , actual time 10495
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 10577  , actual time 10577
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 10656  , actual time 10656
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 10736  , actual time 10736
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 10816  , actual time 10817
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 10895  , actual time 10895
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11055  , actual time 11055
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 11135  , actual time 11135
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11215  , actual time 11215
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 11295  , actual time 11295
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11376  , actual time 11377
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11535  , actual time 11535
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 11615  , actual time 11615
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11696  , actual time 11697
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 11855  , actual time 11855
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 12015  , actual time 12015
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 12175  , actual time 12175
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 12255  , actual time 12255
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 12335  , actual time 12335
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 12415  , actual time 12415
```

Figure 82 Mine Control output

```
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 12495  , actual time 12495
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 12575  , actual time 12575
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 12655  , actual time 12655
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 12735  , actual time 12735
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 12815  , actual time 12815
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 12895  , actual time 12895
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 12975  , actual time 12976
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 13055  , actual time 13055
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 13135  , actual time 13135
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 13295  , actual time 13295
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 18095  , actual time 18095
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 18175  , actual time 18175
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 18255  , actual time 18255
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 18335  , actual time 18335
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 18415  , actual time 18415
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 18495  , actual time 18495
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 18575  , actual time 18575
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 18655  , actual time 18655
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 18735  , actual time 18735
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 18815  , actual time 18815
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 18895  , actual time 18896
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 18976  , actual time 18976
2- ALARM, AIR FLOW LEVEL DANGEROUS, air flow level 2.50, moment 19013 , actual time 19014
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 19055  , actual time 19055
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 19135  , actual time 19135
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 19215  , actual time 19215
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 19295  , actual time 19295
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 19376  , actual time 19376
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 19455  , actual time 19455
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 19536  , actual time 19537
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 19615  , actual time 19616
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 19695  , actual time 19695
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 19775  , actual time 19775
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 19855  , actual time 19855
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 19937  , actual time 19937
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 20015  , actual time 20015
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 20095  , actual time 20096
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 20176  , actual time 20177
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 20255  , actual time 20255
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 20335  , actual time 20335
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 20415  , actual time 20415
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 20496  , actual time 20496
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 20575  , actual time 20575
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 20655  , actual time 20655
```

Figure 83 Mine Control output

```
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 24655  , actual time 24655
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 24736  , actual time 24737
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 24816  , actual time 24816
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 24895  , actual time 24895
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 24975  , actual time 24975
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 25055  , actual time 25055
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.00 , moment 25135  , actual time 25136
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 5.50 , moment 25216  , actual time 25216
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 25296  , actual time 25297
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 25375  , actual time 25375
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 25456  , actual time 25457
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 25536  , actual time 25536
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 25616  , actual time 25616
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 25696  , actual time 25697
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 25776  , actual time 25777
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 25855  , actual time 25855
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 25935  , actual time 25936
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 26016  , actual time 26016
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 26096  , actual time 26096
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 26175  , actual time 26175
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 26255  , actual time 26255
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 26335  , actual time 26335
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 26416  , actual time 26417
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 26496  , actual time 26496
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 26576  , actual time 26576
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 26656  , actual time 26656
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 26735  , actual time 26736
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 26816  , actual time 26816
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.00 , moment 26896  , actual time 26897
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 26976  , actual time 26976
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 27055  , actual time 27055
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 27135  , actual time 27136
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 27215  , actual time 27215
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 27296  , actual time 27298
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 27376  , actual time 27376
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 27456  , actual time 27457
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 27535  , actual time 27536
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 27616  , actual time 27616
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 27696  , actual time 27696
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 27776  , actual time 27776
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 27855  , actual time 27855
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 27935  , actual time 27936
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 28016  , actual time 28016
2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 6.50 , moment 28096  , actual time 28097
```

Figure 84 Mine Control output

```
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.00 , moment 28175  , actual time 28175
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 7.50 , moment 28257  , actual time 28257
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.00 , moment 28336  , actual time 28336
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 28416  , actual time 28416
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 28496  , actual time 28496
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 28575  , actual time 28575
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 28656  , actual time 28657
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 8.50 , moment 28735  , actual time 28735
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.00 , moment 28815  , actual time 28816
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 9.50 , moment 28895  , actual time 28895
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.00 , moment 28976  , actual time 28976
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.50 , moment 29055  , actual time 29055
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.00 , moment 29135  , actual time 29135
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.50 , moment 29215  , actual time 29216
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.00 , moment 29296  , actual time 29296
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.50 , moment 29375  , actual time 29375
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.00 , moment 29456  , actual time 29457
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.50 , moment 29535  , actual time 29535
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.00 , moment 29615  , actual time 29615
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.50 , moment 29695  , actual time 29695
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.00 , moment 29775  , actual time 29775
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 10.50 , moment 29856  , actual time 29856
 2- ALARM, EXPLOSIVE LEVELS OF CH4, ch4 level 11.00 , moment 29935  , actual time 29936
End period

*** END OF TEST minecontrol ***
```

Figure 85 Mine Control output