EDISOFT
DEFENCE & AEROSPACE TECHNOLOGIES

embedded brains

lero THE IRISH SOFTWARE RESEARCH CENTRE

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

jenaoptronik

RTEMS SMP
Ready to Fly

Technical Note
Space Profile
Release 1

CONTENTS

Identification, Copyrights and License

short git commit identifier : 523ffe2

Release	Date	Status	Changes
1	12 April 2019	For review	Initial version of the RTEMS space profile specification, issued for community review.

Contributing author(s)
Sebastian Huber (embedded brains GmbH), Jose Valdez (EDISOFT), Marcel Verhoef (ESA)

Approver	Signature
Marcel Verhoef (ESA)	Marcel Verhoef Digitally signed by Marcel Verhoef Date: 2019.04.12 14:03:27 +02'00'

CHAPTER

ONE

INTRODUCTION

PURPOSE OF THIS DOCUMENT

This document is issued for **PUBLIC REVIEW**.

The due date for receiving comments is **15 MAY 2019**.

In particular we are seeking feedback to:
- the proposed space profile definition and its justification
- (answers to) the open questions identified

Please provide your comments electronically to: Marcel.Verhoef@esa.int
(via simple e-mail, Excel sheet, or marked-up PDF - all are fine)

The final version of this document, that will take any review comments received into account, is expected to appear in June 2019.

Note that the updated document will be used as the *baseline* for the RTEMS qualification project, as performed under ESA Contract No 4000125572. It is therefore very unlikely that any change requests for the space profile can be accommodated after the baseline version of this document has appeared.

If there are any questions on this document, or on the qualification project, please do not hesitate to contact us, either directly by e-mail or via the RTEMS mailing lists.

Finally, we would like to express our thanks to the questionnaire respondents, who have helped to shape this RTEMS space profile proposal.

Note that the respondents comments are reported as-is, albeit anonymised; explicit references to specific products, projects or missions have been removed.

The aim of this activity is to pre-qualify a subset of RTEMS according to ECSS standards (mainly ECSS-E-ST-40C [ECS09] and ECSS-Q-ST-80C Rev.1 [ECS17]).

RTEMS provides a lot of features and a pre-qualification of

- the entire scope is infeasible (time and budget constraints of this activity), and
- the entire scope is not needed (not all features are used).

As with previous pre-qualification activities, we define a space profile which is

- based on needs of the (ESA) space software community,
- based on past applications and usage experiences,
- based on future usage expectations (in particular: multi-core platforms), and
- eases the RTEMS source code tailoring.

This activity will use the current development branch of mainline RTEMS. As a result of this activity changes in mainline RTEMS will be integrated. Since the schedule of this activity and the RTEMS community release process may not correlate it is not yet clear if the baseline for the pre-qualified RTEMS will be a proper RTEMS project release, e.g. RTEMS 5.1.

To get input from users a survey was carried out from 7 March 2019 to 25 March 2019 via a world wide accessible online form. We received 31 responses in total (mostly from ESA member states, 2 from USA). This activity and early results of the survey were presented in an user consultation meeting hosted by ESA on 27 March 2019. The full survey results are presented in section *Survey Questions and Responses*.

CHAPTER

TWO

EXTERNAL DEPENDENCIES

2.1 Compiler

The pre-qualified RTEMS library produced by this activity may have some dependencies on external libraries provided by the compiler (GCC). These external libraries will be used as is (e.g. libgcc). One possible use of such an external library is a 64-bit integer division present in the clock driver initialization. The aim is to avoid such dependencies. The 64-bit integer division is currently the only known dependency.

The integer support functions are moderately complex and adding them to the scope of this activity exceeds our time and budget limits. It is not the job of a real-time operating system kernel to offer basic C support for functions like this:

```
uint64_t div(uint64_t a, uint64_t b)
{
    return a / b;
}
```

Anyway, given the commit history of libgcc2.c, the GCC review procedures, the expert knowledge of the GCC maintainers, and the usage areas and history of this piece of software it can be considered proven in use. The functions provided by libgcc may be also used by applications. The space community should think about a qualified compiler or even a recommended compiler for space applications.

Using an alternative compiler such as LLVM/Clang is out of scope of this activity. In general, there is support available for SPARC/LEON in LLVM/Clang as a result of previous ESA activities.

2.2 Bootloader

The board support package (BSP) may depend on the low-level initialization performed by a bootloader provided by the platform vendor, for example MKPROM2 from Gaisler. A pre-qualification of this software is out of scope of this activity.

2.3 C Library

There will be **no dependency on the Newlib provided libc**. Basic functions like `memcpy()` and `memset()` will be provided by this activity.

2.4 Mathematical Library

The standard mathematical library **libm will not be used** by the pre-qualified RTEMS of this activity. The pre-qualified RTEMS will support the ESA provided MLFS.

CHAPTER

THREE

SPACE PROFILE PROPOSAL

This section gives a list of features, directives, and application configuration options which are included in the space profile. Anything not mentioned here is excluded from the space profile. Excluded means that it is not subject to the pre-qualification activity. Additional functionality can be enabled or added by the end-users, but this must then done under their own responsibility (to complete the qualification for the additional parts). The data package produced by this activity may for example include two libraries for a certain platform, one contains the pre-qualified software components (e.g. librtemsqual) and one contains additional components normally available in mainline RTEMS (e.g. librtemsextra). This allows end-users to select additional components at their own expense in terms of the effort needed to qualify (this is a different approach as taken with Edisoft RTEMS Improvement where features were explicitly removed). For the application configuration defines (CONFIGURE_*) listed here value ranges are not yet defined.

3.1 Platforms

- GR712RC (single-core and dual-core configurations)
- GR740 (single-core and multi-core configurations)

See also *Survey Responses: Processor Architectures*, *Survey Responses: Target Platforms* and *Space Profile Justification: Platforms*.

3.2 Devices

- GPIO, see *Open Issue: GPIO Driver*.
- MIL-STD-1553, see *Open Issue: MIL-STD-1553 Driver*.
- Polled UART single character output with UART initialized by boot loader. This is enough to generate test suite output. See also *Open Issue: UART Driver*.
- SpaceWire, see *Open Issue: SpaceWire Driver*.

See also *Survey Responses: Devices* and *Space Profile Justification: Devices*.

3.3 Locking Protocols and Scheduling

- Transitive Priority Inheritance
- Immediate Ceiling Priority Protocol (ICPP)
- O(m) Independence-Preserving Protocol (OMIP)
- Multiprocessor Resource Sharing Protocol (MrsP)
- Run-time deadlock detection
- Clustered scheduling
- EDF scheduler with one-to-one and one-to-all thread to processor affinity support

See also *Survey Responses: Locking Protocols and Scheduling* and *Survey Responses: Thread Support*.

3.4 C Standard Support

- aligned_alloc()
- calloc()
- malloc() (maybe without support for errno and the deferred free)
- memcpy()
- memset()
- Thread-local Storage (TLS)

See also *Survey Responses: Dynamic Memory*, *Survey Responses: Miscellaneous* and *Space Profile Justification: Dynamic Memory*.

3.5 Initialization

The board support package (BSP) for the platform will initialize the system according to the application configuration. The BSP may rely on low-level initialization performed by a bootloader. See also Initalization Manager.

3.6 Barrier Manager

- CONFIGURE_MAXIMUM_BARRIERS
- rtems_barrier_create()
- rtems_barrier_ident()
- rtems_barrier_release()
- rtems_barrier_wait()

See also *Survey Responses: Barrier Manager* and *Space Profile Justification: Classic API*.

3.7 Clock Manager

- Clock driver providing the clock tick and CLOCK_MONOTONIC
- CONFIGURE_APPLICATION_DOES_NOT_NEED_CLOCK_DRIVER
- CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
- CONFIGURE_MICROSECONDS_PER_TICK
- rtems_clock_get_ticks_per_second()
- rtems_clock_get_ticks_since_boot()
- rtems_clock_get_uptime()

See also *Open Issue: CLOCK_REALTIME*, *Survey Responses: Clock Manager* and *Space Profile Justification: Classic API*.

3.8 Event Manager

- rtems_event_receive()
- rtems_event_send()
- rtems_event_system_receive()
- rtems_event_system_send()

See also *Survey Responses: Event Manager* and *Space Profile Justification: Classic API*.

3.9 Fatal Error Manager

- rtems_fatal()

See also *Survey Responses: Fatal Error Handling* and *Space Profile Justification: Classic API*.

3.10 Interrupt Manager

- CONFIGURE_INTERRUPT_STACK_SIZE
- rtems_interrupt_local_disable()
- rtems_interrupt_local_enable()
- rtems_interrupt_lock_acquire()
- rtems_interrupt_lock_destroy()
- rtems_interrupt_lock_initialize()

- rtems_interrupt_lock_release()

See also *Survey Responses: Interrupt Manager* and *Space Profile Justification: Classic API*.

3.11 Message Manager

- CONFIGURE_MAXIMUM_MESSAGE_QUEUES
- CONFIGURE_MESSAGE_BUFFER_MEMORY
- CONFIGURE_MESSAGE_BUFFERS_FOR_QUEUE
- rtems_message_queue_broadcast()
- rtems_message_queue_create()
- rtems_message_queue_flush()
- rtems_message_queue_get_number_pending()
- rtems_message_queue_ident()
- rtems_message_queue_receive()
- rtems_message_queue_send()
- rtems_message_queue_urgent()

See also *Survey Responses: Message Manager* and *Space Profile Justification: Classic API*.

3.12 Object Manager

- rtems_build_name()

See also *Survey Responses: Classic Objects* and *Space Profile Justification: Classic API*.

3.13 Partition Manager

- CONFIGURE_MAXIMUM_PARTITIONS
- rtems_partition_create()
- rtems_partition_get_buffer()
- rtems_partition_ident()
- rtems_partition_return_buffer()

See also *Survey Responses: Partition Manager* and *Space Profile Justification: Classic API*.

3.14 Rate Monotonic Manager

- CONFIGURE_MAXIMUM_PERIODS
- rtems_rate_monotonic_cancel()
- rtems_rate_monotonic_create()
- rtems_rate_monotonic_deadline()
- rtems_rate_monotonic_get_status()
- rtems_rate_monotonic_ident()
- rtems_rate_monotonic_period()

See also *Survey Responses: Rate Monotonic Manager* and *Space Profile Justification: Classic API*.

3.15 Scheduler Manager

- CONFIGURE_MAXIMUM_PROCESSORS
- CONFIGURE_SCHEDULER_ASSIGNMENTS
- CONFIGURE_SCHEDULER_EDF_SMP
- CONFIGURE_SCHEDULER_TABLE_ENTRIES
- rtems_scheduler_add_processor()
- rtems_scheduler_get_processor()
- rtems_scheduler_get_processor_maximum()
- rtems_scheduler_get_processor_set()
- rtems_scheduler_ident()
- rtems_scheduler_ident_by_processor()
- rtems_scheduler_ident_by_processor_set()
- rtems_scheduler_remove_processor()

See also *Survey Responses: Locking Protocols and Scheduling*, *Survey Responses: SMP Support* and *Space Profile Justification: Classic API*. Please note that `rtems_get_current_processor()` was replaced by `rtems_scheduler_get_processor()` and that `rtems_get_processor_count()` was replaced by `rtems_scheduler_get_processor_maximum()` recently.

3.16 Semaphore Manager

- CONFIGURE_MAXIMUM_MRSP_SEMAPHORES
- CONFIGURE_MAXIMUM_SEMAPHORES
- rtems_semaphore_create()
- rtems_semaphore_ident()

- rtems_semaphore_obtain()
- rtems_semaphore_release()
- rtems_semaphore_set_priority()

See also *Survey Responses: Semaphore Manager* and *Space Profile Justification: Classic API*.

3.17 Task Manager

- CONFIGURE_EXTRA_TASK_STACKS
- CONFIGURE_INIT_TASK_ARGUMENTS
- CONFIGURE_INIT_TASK_ATTRIBUTES
- CONFIGURE_INIT_TASK_ENTRY_POINT
- CONFIGURE_INIT_TASK_INITIAL_MODES
- CONFIGURE_INIT_TASK_PRIORITY
- CONFIGURE_INIT_TASK_STACK_SIZE
- CONFIGURE_MAXIMUM_TASKS
- CONFIGURE_MINIMUM_TASK_STACK_SIZE
- rtems_task_create()
- rtems_task_get_affinity()
- rtems_task_get_priority()
- rtems_task_get_scheduler()
- rtems_task_ident()
- rtems_task_is_suspended()
- rtems_task_iterate()
- rtems_task_restart()
- rtems_task_resume()
- rtems_task_self()
- rtems_task_set_affinity()
- rtems_task_set_priority()
- rtems_task_set_scheduler()
- rtems_task_start()
- rtems_task_suspend()
- rtems_task_wake_after()

See also *Survey Responses: Locking Protocols and Scheduling*, *Survey Responses: Thread Support*, *Survey Responses: Task Manager* and *Space Profile Justification: Classic API*.

3.18 Timer Manager

- CONFIGURE_MAXIMUM_TIMERS
- rtems_timer_cancel()
- rtems_timer_create()
- rtems_timer_fire_after()
- rtems_timer_ident()
- rtems_timer_initiate_server()
- rtems_timer_reset()
- rtems_timer_server_fire_after()

See also *Survey Responses: Timer Manager* and *Space Profile Justification: Classic API*.

3.19 Extensions Manager

- CONFIGURE_INITIAL_EXTENSIONS
- CONFIGURE_MAXIMUM_USER_EXTENSIONS
- rtems_extension_create()
- rtems_extension_ident()

See also *Survey Responses: Extensions Manager* and *Space Profile Justification: Classic API*.

CHAPTER

FOUR

SPACE PROFILE JUSTIFICATION

4.1 Platforms

4.1.1 Included

The *survey results confirm* that the LEON3/LEON4 Gaisler platforms GR712RC and GR740 selected for this activity are the most interesting for the respondents of the survey.

4.1.2 Excluded

The questionnaire identified also the need to qualify RTEMS on other platforms (single-core LEONs UT699/UT700, ARM, RISC-V, etc). Even though these platforms cannot be targeted in the context of this activity due to resource constraints, however, the infrastructure created will allow to easily add support for them, e.g. as part of future activities or by community contribution.

4.2 Devices

4.2.1 Included

Included devices are GPIO, MIL-STD-1553, UART and SpaceWire. See also *Open Issue: Devices Driver Interfaces*.

4.2.2 Excluded

Excluded devices are CAN, I2C, PCI, SPI and Ethernet due to a lack of user interest or time and budget constraints. To avoid duplicated work in space projects it is recommended to add more device drivers in follow up activities to a common pool which can be used by space missions. This activity will do the ground work to set up the infrastructure for this.

4.3 Programming and Parallel Languages

4.3.1 Included - C

C is supported including thread-local storage (TLS) support. Most of the C standard library support is excluded.

4.3.2 Included - uPython

For compatibility with previous uPython activities carried out by ESA (see https://essr.esa.int/project/micropython-for-leon), support for it will be included in the space profile. However, not all features will be supported, e.g. task notes were removed in RTEMS 5.1. The rtems_*_delete() directives are not in the space profile.

4.3.3 Excluded - Ada

In mainline RTEMS 5.1 the process to build a GCC with Ada support was substantially simplified since now all header files required by the Ada runtime library are included in Newlib. So, the GCC can be built in one step without the need to build RTEMS first. Also the task variables were replaced by thread-local storage (TLS) variables to support SMP configurations. Tool chains with Ada support are regularly built for ARM, PowerPC, RISC-V and SPARC targets by RTEMS community members. However, the Ada support depends on the excluded POSIX API. It would be possible to pre-qualify the additional components in a parallel or follow up activity. This activity should avoid dependencies on the POSIX signals (e.g. `pthread_sigmask()`, `pthread_kill()`, and `sigaction()`) since their implementation in RTEMS has a questionable quality.

4.3.4 Excluded - C++, Java, Go, Rust and LUA

C++ may work out of the box, however, at your own risk. Most parts of the standard C++ library will require the components outside the pre-qualified RTEMS library.

The demand for Java, Go, and Rust was quite weak. LUA was mentioned in a remark. In general, programming language support can be added in follow up projects. All relevant synchronization primitives are included in the space profile. There is just a lack of time and budget to cover all API variants.

4.3.5 Excluded - OpenMP

There was some interest in OpenMP. The OpenMP in GCC is well supported in mainline RTEMS and on par with the Linux support. Critical synchronization primitives such as the OpenMP barriers use the Linux code based on Futexes as is. The OpenMP support in GCC consists of three parts:

1. The code generation through the compiler.
2. The libgomp runtime support library.

3. The operating system services required by libgomp.

For a qualified application all three parts must be taken into account. Providing only part three in this activity would still require at least a pre-qualification of libgomp which is completely out of scope of this activity. The operating system services required by libgomp are mainly POSIX API functions. OpenMP would be the only valid reason to include these POSIX API functions in the space profile. To focus on the overall quality of the pre-qualification activity it was decided to not include OpenMP unless the project progress is exceptionally good.

4.3.6 Excluded - MTAPI

There was no user interest in MTAPI. It will not be included in the space profile.

4.4 Dynamic Memory

4.4.1 Included

The *survey results* suggest that dynamic memory allocation during system initialization is highly desired. Therefore the space profile will include the `aligned_alloc()`, `calloc()` and `malloc()` functions. It will probably not support the setting of `errno` to `ENOMEM` in case of an allocation failure. It will probably also not support the deferred free, since the `free()` function is excluded in the space profile.

4.4.2 Excluded

Excluded are the free of allocated memory and reallocations.

4.5 Classic API

4.5.1 Included

The following RTEMS Classic API managers are included in parts in the space profile:

- *Barrier Manager*
- *Clock Manager*
- *Event Manager*
- *Fatal Error Manager*
- *Interrupt Manager*
- *Message Manager*
- *Object Manager*
- *Partition Manager*
- *Rate Monotonic Manager*

- *Scheduler Manager*
- *Semaphore Manager*
- *Task Manager*
- *Timer Manager*
- *Extensions Manager*

4.5.2 Open - IO Manager

There was some interest in the IO Manager, see also *Open Issue: IO Manager*.

4.5.3 Excluded

Explicitly excluded are

- all object delete directives,
- all diagnostic directives, e.g. `rtems_rate_monotonic_get_statistics()`,
- the `rtems_clock_get_tod()` directive,
- the `rtems_timer_fire_when()` directive,
- the `rtems_timer_server_fire_when()` directive, and
- `rtems_task_wake_when()` directive.

With respect to the `rtems_*_when()` and `rtems_clock_get_tod()` directives, see *Open Issue: CLOCK_REALTIME*.

The Signal Manager is excluded due to no strong interest from the user side and the questionable implementation. Signals are called as a side effect of thread dispatching. No application code should run in this area. There is a possibility of infinite recursion since signal handlers may block leading to a thread dispatch.

The Region Manager is excluded due to little interest from the user side. Its use is not recommended in general as long as it is implemented as a first-fit allocator.

The Dual Ported Memory Manager is excluded since is lacks a proper use case.

A lot of diagnostic and debug support is excluded due to time and budget constraints. These services can still be used during application development. They just have to be removed in the flight software configuration.

There will be no support for asymmetric multiprocessing (RTEMS_MULTIPOCESSING). It was not included in the Edisoft RTEMS Improvement as well. It is rarely tested by the RTEMS community work flow. During the massive code rewrites to support SMP things may have broken the RTEMS_MULTIPOCESSING support in subtle ways. The current status is that it is compile clean. The implementation exploits that the MPCI thread has the highest priority and is non-preemptive. This exploit breaks in SMP configuration.

4.6 Other Features

4.6.1 Open - CLOCK_REALTIME

There is some interest in CLOCK_REALTIME support, however, it is not clear what CLOCK_REALTIME should be. See also *Open Issue: CLOCK_REALTIME*.

4.6.2 Excluded - Arbitrary Thread to Processor Affinity

There was some interest in support for **arbitrary** thread to processor affinities. However, this is only supported in a proof-of-concept scheduler in RTEMS and a production quality implementation is quite complex, see [GCB13] and [CGB14].

4.6.3 Excluded - Reclamation of Dynamic Memory or Objects

There was little interest in the support to free dynamic memory or delete objects. There was some interest in thread deletion in error handling situations. This use case can be also addressed through a thread restart. There will be no `free()` function or ability to delete objects (e.g. `rtems_task_delete()`, `rtems_semaphore_delete()`, etc.).

4.6.4 Excluded - POSIX API

There was little user interest in POSIX API features. It will not be included in the space profile.

4.6.5 Excluded - File Systems

There was some interest in file systems. They are excluded in this activity due to time and budget constraints.

4.6.6 Excluded - Network Stack

A network stack will be not included in the space profile. It may still be used during application development. The goal is to provide an add-on library so that you have access to the full RTEMS feature set during development.

CHAPTER

FIVE

COMPARISON TO EDISOFT RTEMS IMPROVEMENT

The following subsections present a summary of the C Library, Application Configuration and the Managers that are present in RTEMS Improvement. The **bolded** and surrounded with hash tags, (e.g. #**items**#) are the ones that are **not present** in RTEMS SMP Space Profile.

The purpose of this section is to show users of RTEMS Improvement, what will change in case they consider migrating towards the new RTEMS SMP Space profile.

5.1 C Library

In the following table is presented the necessary auxiliary C Library functions for RTEMS Improvement.

Table 1: RTEMS Improvement C Library

Function
memcpy()
memset()
#**_mktm_r()**#
#**strncmp()**#

5.2 Application Configuration

In the following table, the Configuration variables for RTEMS Improvement are presented. For some variables the default value is presented (i.e the value assumed by RTEMS, when the user does not define one) and the maximum value (a compile warning will be shown in such a case).

Table 2: RTEMS Improvement Application Configuration

Variable	Default	Max
Time Management:		
CONFIGURE_MICROSECONDS_PER_TICK	10000	-
#**CONFIGURE_TICKS_PER_TIMESLICE**#	50	-
Object Management:		

Continued on next page

Table 2 – continued from previous page

Variable	Default	Max
CONFIGURE_MAXIMUM_TASKS	0 (a warning will be generated, since this value should be redefined to be at least 1, to account for the Init task)	64
CONFIGURE_MAXIMUM_TIMERS	0	64
CONFIGURE_MAXIMUM_SEMAPHORES	0	256
CONFIGURE_MAXIMUM_MESSAGE_QUEUES	0	256
CONFIGURE_MAXIMUM_PERIODS	0	64
CONFIGURE_MAXIMUM_USER_EXTENSIONS	0	16
#CONFIGURE_SEM_SECURE_BLOCK_SUSPEND#	0	-
Device Driver Management:		
#CONFIGURE_HAS_OWN_DEVICE_DRIVER_TABLE#	Not Defined	-
#CONFIGURE_MAXIMUM_DRIVERS#	number of entries in Devices_drivers array	-
CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER	Not Defined	-
#CONFIGURE_APPLICATION_NEEDS_STUB_DRIVER#	Not Defined	-
#CONFIGURE_APPLICATION_EXTRA_DRIVERS#	Not Defined	-
Initialization Tasks Table Configuration:		
#CONFIGURE_RTEMS_INIT_TASKS_TABLE#	Defined	-
#CONFIGURE_HAS_OWN_INIT_TASK_TABLE#	Not defined (use RTEMS default)	-
CONFIGURE_INIT_TASK_NAME	rtems_build_name (‘U’, ‘I’, ‘1’, ‘’)	-
CONFIGURE_INIT_TASK_STACK_SIZE	RTEMS_MINIMUM _STACK_SIZE	-
#RTEMS_MINIMUM_STACK_SIZE#	1024 * 4	-
CONFIGURE_INIT_TASK_PRIORITY	1	-
CONFIGURE_INIT_TASK_ATTRIBUTES	RTEMS _DEFAULT _ATTRIBUTES	-
#RTEMS_DEFAULT_ATTRIBUTES#	0x00000000 (meaning: RTEMS_NO _FLOATING_POINT and RTEMS _APLICATION _TASK)	-
CONFIGURE_INIT_TASK_ENTRY_POINT	Init	-
CONFIGURE_INIT_TASK_INITIAL_MODES	RTEMS_NO _PREEMPT	-
CONFIGURE_INIT_TASK_ARGUMENTS	0	-
Extra parameters:		
#CONFIGURE_HAS_OWN_CONFIGURATION_TABLE#	Not defined	-
#CONFIGURE_INTERRUPT_STACK_MEMORY#	RTEMS _MINIMUM _STACK _SIZE	-
#RTEMS_MINIMUM_STACK_SIZE#	1024 * 4	-

Continued on next page

Table 2 – continued from previous page

Variable	Default	Max
#CONFIGURE_EXECUTIVE_RAM_WORK_AREA#	NULL	-
CONFIGURE_MESSAGE_BUFFER_MEMORY	0	-
#CONFIGURE_MEMORY_OVERHEAD#	0	-
CONFIGURE_EXTRA_TASK_STACKS	0	-
Fatal Error Handling:		
#CONFIGURE_HAS_OWN_APP_SAFE_STATE_HANDLER#	Not Defined	-

5.3 RTEMS Managers

In the following table, the RTEMS Managers and the respective directives selected for RTEMS Improvement are presented.

Table 3: RTEMS Improvement Managers

Manager	Functions
Initialization Manager:	**#rtems_initialize_executive_early()#***
	#rtems_initialize_executive_late()#*
	rtems_shutdown_executive()
	*Remarks: These directives were replaced by rtems_initialize_executive()
Task Manager:	rtems_task_create()
	rtems_task_ident()
	rtems_task_start()
	rtems_task_restart()
	#rtems_task_delete()#
	rtems_task_suspend()
	rtems_task_resume()
	rtems_task_is_suspended()
	rtems_task_set_priority()
	#rtems_task_mode()#
	#rtems_task_set_note()#
	#rtems_task_get_note()#
	rtems_task_wake_after()
	#rtems_task_wake_when()#
	#rtems_task_variable_add()#
	#rtems_task_variable_get()#
	#rtems_task_variable_delete()#
#Interrupt Manager#:	**#rtems_interrupt_catch()#**
	#rtems_interrupt_disable()#
	#rtems_interrupt_enable()#
	#rtems_interrupt_flash()#
	#rtems_interrupt_is_in_progress()#
	#rtems_interrupt_is_masked()#
	#rtems_mask_interrupt()#
	#rtems_unmask_interrupt()#

Continued on next page

Table 3 – continued from previous page

Manager	Functions
Clock Manager:	**#rtems_clock_set()#**
	#rtems_clock_get()#
	rtems_clock_get_uptime()
	#rtems_clock_set_nanoseconds_extension()#
	#rtems_clock_tick()#
Timer Manager:	rtems_timer_create()
	rtems_timer_ident()
	rtems_timer_cancel()
	#rtems_timer_delete()#
	rtems_timer_fire_after()
	#rtems_timer_fire_when()#
	rtems_timer_initiate_server()
	rtems_timer_server_fire_after()
	#rtems_timer_server_fire_when()#
	rtems_timer_reset()
Semaphore Manager:	rtems_semaphore_create()
	rtems_semaphore_ident()
	#rtems_semaphore_delete()#
	rtems_semaphore_obtain()
	rtems_semaphore_release()
	#rtems_semaphore_flush()#
Message Queue Manager:	rtems_message_queue_create()
	rtems_message_queue_ident()
	#rtems_message_queue_delete()#
	rtems_message_queue_send()
	rtems_message_queue_urgent()
	rtems_message_queue_broadcast()
	rtems_message_queue_receive()
	rtems_message_queue_get_number_pending()
	rtems_message_queue_flush()
Event Manager:	rtems_event_send()
	rtems_event_receive()
#I/O Manager#:	**#rtems_io_register_driver()#**
	#rtems_io_initialize()#
	#rtems_io_open()#
	#rtems_io_close()#
	#rtems_io_read()#
	#rtems_io_write()#
	#rtems_io_control()#
#Error Manager#:	**#rtems_fatal_error_occurred()#**
	#rtems_error_report()#
	#rtems_error_get_latest_non_fatal_by_offset()#
	#rtems_error_get_latest_fatal_by_offset()#
Rate Monotonic Manager:	rtems_rate_monotonic_create()
	rtems_rate_monotonic_ident()

Continued on next page

Table 3 – continued from previous page

Manager	Functions
	rtems_rate_monotonic_cancel()
	#rtems_rate_monotonic_delete()#
	rtems_rate_monotonic_period()
	rtems_rate_monotonic_get_status()
	rtems_rate_monotonic_deadline()
	#rtems_rate_monotonic_get_deadline_state()#
	#rtems_rate_monotonic_execution_time()#
User Extensions Manager:	rtems_extension_create()
	rtems_extension_ident()
	#rtems_extension_delete()#

CHAPTER

SIX

OPEN ISSUES

6.1 CLOCK_REALTIME

The survey results suggest that there is an interest in supporting CLOCK_REALTIME. This clock must be specified first. It raises the question how the time synchronization should be done (e.g. Network Time Protocol)? How is the leap second handling performed? How do you get the initial time at system start?

6.2 Fatal Error Handling

The API for fatal error handling in mainline RTEMS and Edisoft RTEMS Improvement diverged substantially. There are no plans to integrate the Edisoft RTEMS Improvement fatal error handling support in mainline RTEMS.

6.3 IO Manager

Is this poorly designed API really used for device drivers in mission critical software? Which problem does it solve? Do you really want to use global variables for the major and minor device numbers to access your drivers? Do you really want to read and write to devices via a void pointer interface with absolutely no static type checking? This API offers no chance for static analysis tools to produce anything useful.

6.4 Interrupt Controller Interface

The interrupt controller interface must be specified. This includes interrupt handler installation and removal, disabling and enabling of specific interrupts, setting of interrupt priorities, etc.

6.5 GPIO Driver

The *survey results* suggest that some users would like to have an GPIO driver included in the space profile. The interface and feature set of such a driver must be specified. GPIO drivers are available in mainline RTEMS based on the Gaisler driver manager.

6.6 MIL-STD-1553 Driver

The *survey results* suggest that some users would like to have an MIL-STD-1553 driver included in the space profile. The interface and feature set of such a driver must be specified. MIL-STD-153 drivers are available in mainline RTEMS based on the Gaisler driver manager and also previous ESA activities.

6.7 UART Driver

The *survey results* suggest that some users would like to have an UART driver included in the space profile. The interface and feature set of such a driver must be specified. The standard UART drivers of RTEMS use the Termios framework. This seems to be a bit too complex and complicated for the intended criticality category. Things to consider are the device settings (baud, bits per word, parity, stop bits), how to set them, flow control yes/no/software/hardware, a potential line mode, timeout handling, character conversion, buffering, interrupt or polled mode, DMA support, etc.

6.8 SpaceWire Driver

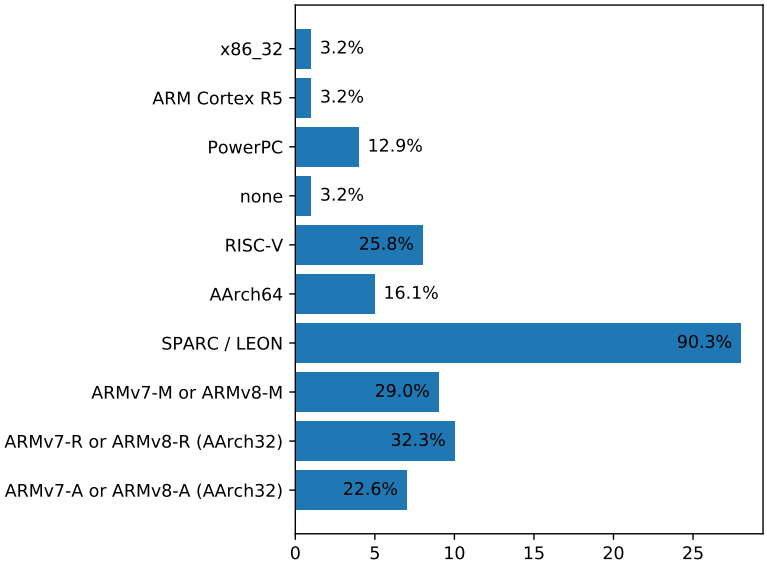
The *survey results* suggest that some users would like to have an SpaceWire driver included in the space profile. The interface and feature set of such a driver must be specified. SpaceWire drivers are available in mainline RTEMS based on the Gaisler driver manager and also previous ESA activities.

CHAPTER

SEVEN

SURVEY QUESTIONS AND RESPONSES

7.1 Processor Architectures

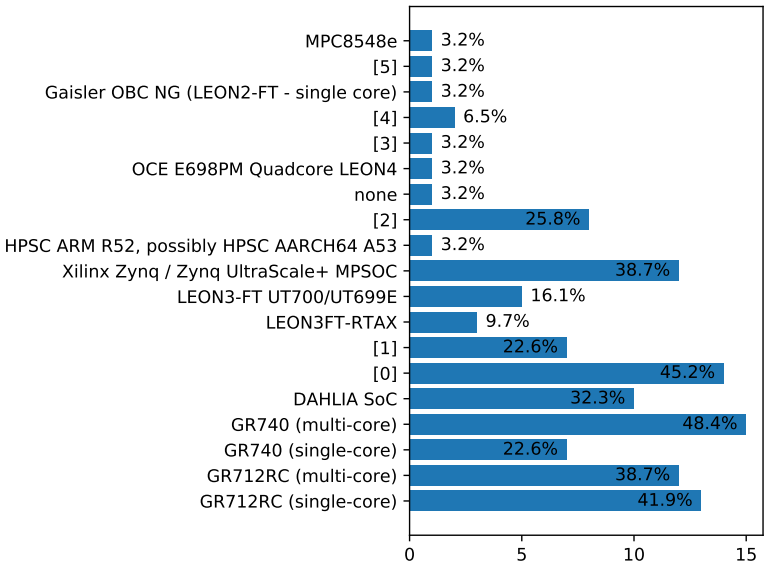


Collected remarks from questionnaire respondents:

- “LEON3/LEON4 is the most important and near term. ARM targets are longer term, RISC-V is speculative at this point.”
- “None (we are tool developers).”
- “Mass Memory applications”
- “Gaisler GR740 processor”
- “PowerPC is desirable, not required”
- “These are computer architectures we are looking at for future projects. The availability of RTEMS on such architectures would be welcome as an additional solution. This does not necessarily imply at this stage that if RTEMS SMP would be supported on such architecture, than it would be the selected solution.”
- “Cortex-Rs added due to DAHLIA.”

See also *Space Profile Proposal: Platforms*.

7.2 Target Platforms



Legenda to other target architectures identified (see figure above):

0. Time-space partitioning kernel (AIR, PikeOS, XtratuM)
1. Aeroflex Gaisler UT699 (LEON3FT single-core)
2. Microchip / ATMEL SAMV71 (ARM Cortex-M7 single-core)
3. ARM Cortex-M3, also using ERC32 for legacy projects
4. NXP QorIQ (e.g. BAE RAD5500, TeleDyne E2V P5020, TeleDyne E2V LS1046)
5. LEON3 would be seen favourably mostly to have a single and most recent version of RTEMS (5) qualifiable for all LEON3 - LEON4 based platforms

Collected remarks from questionnaire respondents:

- “We are tool suppliers, we will support any platform requried.”
- “Our main platform today is LEON2, as it provides sufficient performances.”
- “Again, these are processors / TSP RTOS on which the availability of RTEMS SMP would be one among other possible solutions. It does not imply that if RTEMS SMP would support those, then it would be selected.”
- “These are platforms that are either used, or going to be used in missions and R&D projects.”

See also *Space Profile Proposal: Platforms*.

7.3 Devices

Table 1: Devices

Feature	MUST	SHOULD	COULD	WON'T
CAN	8	8	10	5
GPIO	24	5	1	1
I2C	3	6	13	9
MIL-STD-1553	18	7	0	6
PCI	4	6	9	12
SpaceWire	25	3	2	1
SPI	11	8	8	4
UART	22	6	0	3
Ethernet	7	9	13	2

Collected remarks from questionnaire respondents:

- "Need GPIO for very high speed and low overhead tracing for WCET analysis."
- "We write all drivers ourselves."
- "UART is mainly for debugging (disabled in flight configuration)"
- "Derived from our baseline DHS HW architecture."
- "1553, SpW, UART, GPIO present in most architectures. ETH at least as debug link. SPI and PCI as second priority. I2C present in some architectures but not all."
- "Ethernet functional for on-ground testing purpose. Not necessary in flight."

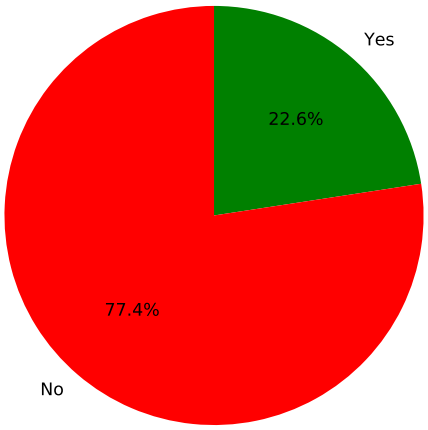
See also *Space Profile Proposal: Devices*.

7.4 Dynamic Memory

Table 2: Dynamic Memory

Feature	MUST	SHOULD	COULD	WON'T
Do you want to allocate memory during initialization?	17	7	4	3
Do you want to allocate memory after initialization?	2	5	17	7
Do you want to free allocated memory?	0	2	10	19
Do you need a scalable SMP memory allocator, e.g. FreeBSD UMA or jemalloc?	0	2	15	14

Do you expect problems due to memory fragmentation?



Collected remarks from questionnaire respondents:

- "No dynamic memory needs for tracing for WCET analysis."
- "No dynamic memory allocation in our SW architecture, but rather custom static memory allocator "
- "Our standards ALLOW, but discourage, memory allocation at initialization, and dissallow memory allocation after initialization with exceptions permited in some very constrained cases. In some of our missions, we have removed malloc from the library so it can't be used."
- "For some cat. C applications, memory allocation after initialisation or free of allocated memory could add flexibility (e.g., partial reconfiguration / reload)"
- "To my knowledge, in Cat-C and B, we don't allocate after initialization, and we never

free; so there's no need for scaleable SMP allocator, and there can be no fragmentation."

See also *Space Profile Proposal: C Standard Support*.

7.5 Thread Support

Table 3: Thread Support

Feature	MUST	SHOULD	COULD	WON'T
Do you want to delete threads?	5	4	5	17
Do you want to delete other threads than the executing thread?	4	2	6	19
Do you want to restart threads?	4	6	10	11
Do you want to restart other threads than the executing thread?	4	4	13	10
Do you want to use a one-to-one thread to processor affinity?	14	7	5	5
Do you want to use arbitrary thread to processor affinity?	5	2	15	9

Why do you want to use thread to processor affinity?

- "Prevent jitter in switching time that could appear as a thread switches CPUs. "
- "Predictability"
- "potential workload balancing"
- "To optimize latencies and minimize overhead of thread migration when not needed."
- "Determinism and simplification of design."
- "Should not be used!!!. Makes WCET analysis extremely hard and very pessimistic, no Scheduling solution for thread migration really exists. Does provide negative value for schedulability analysis and WCET analysis."
- "To best segregate the functions"
- "Simplification of the schedulability analysis; Processor cache ussage"
- "For more control"
- "To ensure realtime performance of critical tasks, and to simplify schedulability analysis"
- "Pre-defined static allocation of threads to processor only"
- "For execution determinism"
- "yes"
- "For real-time guarantees and minimizing inter-core interference"

- “Real-time processing constraints”
- “to formally prove hard real-time constraints to be met”
- “The ability to control affinity MAY be useful in some circumstances for functions that are required to be highly responsive.”
- “Separate applications with different real-time properties”
- “Static assignment preferred in priority.”
- “Deterministic execution timing”
- “System control”
- “Some missions work on AMP-like configurations - e.g. allocating I/O task on a single core, processing tasks on other”
- “Basic feature found in other OS.”
- “Reduce complexity of the schedulability analysis”

Collected remarks from questionnaire respondents:

- “Restart for recovery only, use on single-core processors must be possible.”
- “Our SW architecture does not require stopping, restarting or suspending tasks”
- “Difficult to have a single reply. Very different strategy could be possible according to a mission cat. B institutional or commercial or low-cost or cat. C.”
- “The suspend/restart is used in missions for e.g. diagnosis tasks. The deletion of a task is allowed in missions only during initialization - if unexpected failures occur and a retry needs to be attempted.”

See also *Space Profile Proposal: Locking Protocols and Scheduling* and *Space Profile Proposal: Task Manager*.

7.6 Locking Protocols and Scheduling

Table 4: Locking Protocols and Scheduling

Feature	MUST	SHOULD	COULD	WON'T
Do you want to use lock-free algorithms?	5	11	11	4
Do you want to use Futex based synchronization?	0	10	16	5
Do you want to use the transitive priority inheritance protocol?	4	11	13	3
Do you want to use the OMIP locking protocol()?	1	6	17	7
Do you want to use the priority ceiling locking protocol?	6	10	11	4
Do you want to use the MrsP locking protocol?	2	5	15	9
Do you want a run-time deadlock detection in mutex lock operations?	5	10	12	4
Do you want to use clustered scheduling?	4	3	17	7
Do you want to use the Earliest Deadline First (EDF) scheduler?	6	4	10	11
Do you want to use the Constant Bandwidth Server (CBS) scheduler?	0	5	15	11
Do you want to perform a schedulability analysis?	17	6	8	0

How do you want to perform a schedulability analysis?

- "Any"
- "This is an open question."
- "Automatic analysis based on simple methods and based on SIMPLE scheduling models."
- "Pen and Paper"
- "execution analysis"
- "By using tools (AbsInt)"
- "On a per-processor basis"
- "Not yet defined. Hypothesis is to use a custom thread-based tool that analyses individual cores for schedulability and design review for multicore aspects."
- "SW prototyping and timing measurement"
- "Now we are modelling the system (e.g. in MAST); schedulability analysis based on static code analysis would be very welcome."

- "Preferably - code based static analysis, or code model extraction for external tools (like MAST etc.)"
- "Response Time Analysis according to MrsP"
- "I would like to know that too ;-)"
- "On a critical subset of SW, the minimum is to be able to feed analytical equations with results of on-target measurements. More advanced techniques according to RTEMS features that are used (level of non-determinisms. . . .) and available tools"
- "Our application has few interrupt driven periodic tasks and some few background tasks. The interdependence is limited. With knowledge of the design the timing can be analysed with rather naïve methods."
- "Own tools"
- "Per ECSS-E40 compliance (5.8.3.11), schedulability analysis techniques are mandatory. The "how" is an open question!"
- "As simply as possible ;-)"

Collected remarks from questionnaire respondents:

- "We do not yet have experience with SMP in real time systems. Over time, as we gain experience, we expect to explore some of these options. For now I just answered "could", but we are not actively studying these features right now. "
- "Task model should be so simple that would make any schedullability analysis trivial. For example, simple fixed priority worst-case response time analysis. The only open issue is to determine WCET."
- "In our SW architecture, to maximise reuse from existing code base and to implement core affinity based priority scheduling on multi-core processor by maximising determinism"
- "EDF and CBS could be used in a system without task priorities."
- "According to the different applications I could have different profiles of desired mechanisms."
- "We were unsure what the "transitive" term referred to (couldn't find it in docs.rtems.org)."
- "Difficult to answer this generally. Depending on the application, a different set of locking/scheduling will be necessary. And MUST everywhere is not a good answer either. . . "

See also *Space Profile Proposal: Locking Protocols and Scheduling* and *Space Profile Proposal: Task Manager*.

7.7 Programming Languages and Libraries

Table 5: Programming Languages and Libraries

Feature	MUST	SHOULD	COULD	WON'T
Do you want to use Ada?	3	3	5	20
Do you want to use C++?	6	8	9	8
Do you want to use Java?	0	0	4	27
Do you want to use the Go programming language?	1	0	2	28
Do you want to use (micro) Python?	2	1	17	11
Do you want to use Rust?	0	0	11	20
Do you want to use the OpenMP?	2	7	13	9
Do you want to use the Gaisler MTAPI?	0	1	20	10
Do you want to use the EMBB MTAPI?	0	0	21	10

Collected remarks from questionnaire respondents:

- "C"
- "C is the only hard requirement here, C++ is a possibility for integrating existing code. Python and Rust are more aligned with research than actual space requirements. OpenMP applies to HPSC. "
- "Ada and C are easy to analyse. C++ is a nightmare for compiler verification and testing. All other languages do not have a pedigree yet for critical software and there is no ecosystem for verification. "
- "Mainly C language and Ada"
- "VHDL"
- "Python for test only (e.g. HW boards tests)"
- "C and Assembler are the baseline. Gaisler support building blocks could be useful to reduce cost of our SW development "
- "LUA is our preferred language for OBCP's. Why: LUA is highly modular."
- "On multi core hardware we intend to manually allocate software applications to the processors. Most probably the applications on the different cores will be developed by different contractors."

7.8 Miscellaneous

Table 6: Miscellaneous

Feature	MUST	SHOULD	COULD	WON'T
Do you want to use CLOCK_REALTIME?	13	6	10	2
Do you want to use CLOCK_MONOTONIC?	9	10	9	3
Do you want to use file systems? If yes, remark which in the comments below.	7	2	10	12
Do you want to use FILE objects?	4	1	14	12
Do you want to use thread-local storage (TLS)?	3	5	15	8
Do you want to use functions which require errno?	5	6	13	7
Do you want to use the C locale support?	2	2	7	20
Do you want to use functions which require Newlib struct _reent?	3	3	14	11
Do you want to use the Gaisler driver manager?	7	4	11	9

Collected remarks from questionnaire respondents:

- "Need RFS and a good flash filesystem"
- "Will use IMFS and RFS. Could use flash file systems"
- "in memory file system + custom file system for EEPROM, Flash"
- "Guaranteeing timing behaviour requires limiting dynamic features and use of complex libraries like file systems. Our objective is to provide accurate and tight schedulability analysis and WCET analysis, the times marked Won't in this table don't help"
- "No POSIX or other support packages required by our SW architecture because not in our baseline. Gaisler drive manager would be useful to reduce development cost"
- "RFS and JFFS"
- "FS: RFS, TFTP, JFFS2 "
- "File system must support FLASH based Mass Memory"
- "Even if a monotonic clock is used, some provision for setting the time is necessary. It also must be set during testing and validation, sometimes frequently. So these capabilies must be provided for even if monotonic is used."
- "Not enough internal feedback on the last 4 questions to reply. Left as "Could" for the moment"
- "errno-wise, e.g. the MLFS will set it (qualified mathematical library)."

- "a read-only filesystem only. Drvmgr, I would like to use non-qualified drivers "out of the box" also and to get updates when drivers are updated (only certain parts of the drvmgr needed)."

7.9 Barrier Manager

Table 7: Barrier Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_barrier_create()	7	4	2	18
rtems_barrier_ident()	2	5	6	18
rtems_barrier_release()	7	4	2	18
rtems_barrier_wait()	7	4	2	18

Collected remarks from questionnaire respondents:

- "If you want to use the barrier manager, why would you not need some or all of these functions?"
- "This API was used in at least one mission."

See also *Space Profile Proposal: Barrier Manager*.

7.10 Clock Manager

Table 8: Clock Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_clock_get_ticks_per_second()	19	5	1	6
rtems_clock_get_ticks_since_boot()	17	5	3	6

Collected remarks from questionnaire respondents:

- "We use the clock manager through a software bus / executive layer on top of RTEMS. I'm not sure which of these functions that layer may use, but I would want them to be available."
- "Note that more APIs are used than mentioned here - specifically, the complete list (obtained from analysis of existing mission code bases) includes: rtems_clock_get_uptime, rtems_clock_set, rtems_clock_tick, rtems_clock_get_options, rtems_clock_get_ticks_since_boot, rtems_clock_get, rtems_clock_get_ticks_per_second"

See also *Space Profile Proposal: Clock Manager*.

7.11 Event Manager

Table 9: Event Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_event_receive()	15	6	3	7
rtems_event_send()	15	6	3	7
rtems_event_system_receive()	2	4	12	13
rtems_event_system_send()	2	4	12	13

Collected remarks from questionnaire respondents:

- "not familiar with rtems_event_system apis"
- "could be useful for UML modeling (execution framework)"
- "Inheritance from our legacy SW architecture "
- "The event manager is not absolutely necessary. It falls into the could category."
- "system events were not seen in any mission codebase so far."

See also *Space Profile Proposal: Event Manager*.

7.12 Interrupt Manager

Table 10: Interrupt Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_interrupt_local_disable()	15	8	3	5
rtems_interrupt_local_enable()	15	8	3	5
RTEMS_INTERRUPT_LOCK_DECLARE()	7	5	12	7
RTEMS_INTERRUPT_LOCK_DEFINE()	7	5	12	7
RTEMS_INTERRUPT_LOCK_MEMBER()	6	5	13	7
rtems_interrupt_lock_acquire()	10	4	11	6
rtems_interrupt_lock_acquire_isr()	9	5	11	6
rtems_interrupt_lock_destroy()	5	3	14	9
rtems_interrupt_lock_initialize()	9	4	11	7
rtems_interrupt_lock_release()	10	5	10	6
rtems_interrupt_lock_release_isr()	9	6	10	6

Collected remarks from questionnaire respondents:

- "We only use rtems_interrupt_catch()"
- "Planned to be used for critical sections definitions and interrupts handling on multi-core processor"
- "cache freezing during interrupts should be supported (which wasn't in RTEMS 4.8.1 on LEON2 with Gaisler BSP)"
- "The interrupt-local disable/enable cannot be used for creating critical sections anymore (in the SMP world). But it still has value if e.g. a single core is in control of I/O and wants to manage specific interrupt lines."

See also *Space Profile Proposal: Interrupt Manager*.

7.13 IO Manager

Table 11: IO Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_io_close()	6	7	2	16
rtems_io_control()	6	7	2	16
rtems_io_initialize()	6	7	2	16
rtems_io_open()	6	7	2	16
rtems_io_read()	6	7	2	16
rtems_io_register_driver()	7	6	2	16
rtems_io_write()	6	7	2	16

Collected remarks from questionnaire respondents:

- "don't know yet"
- "This allows mission-specific driver/API creation."

7.14 Message Manager

Table 12: Message Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_message_queue_broadcast()	7	4	5	15
rtems_message_queue_create()	18	4	1	8
rtems_message_queue_delete()	8	4	5	14
rtems_message_queue_flush()	12	6	1	12
rtems_message_queue_get_number_pending()	11	8	1	11
rtems_message_queue_ident()?	11	3	6	11
rtems_message_queue_receive()	18	4	1	8
rtems_message_queue_send()	18	4	1	8
rtems_message_queue_urgent()	10	4	5	12

Collected remarks from questionnaire respondents:

- "Creation only at startup"
- "The "delete" is in SHOULD, because generally these are resources that are never "freed" in mission codebases."
- "grspw_pkt driver uses messages from ISR to work-task/user. The driver could be simplified to remove the message-queue?"

See also *Space Profile Proposal: Message Manager*.

7.15 Classic Objects

Table 13: Classic Objects

Feature	MUST	SHOULD	COULD	WON'T
rtems_build_name()	12	9	8	2

See also *Space Profile Proposal: Object Manager*.

7.16 Partition Manager

Table 14: Partition Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_partition_create()	8	4	3	16
rtems_partition_delete()	2	2	6	21
rtems_partition_get_buffer()	7	4	4	16
rtems_partition_ident()	4	5	5	17
rtems_partition_return_buffer()	7	4	4	16

Collected remarks from questionnaire respondents:

- "don't know yet"

See also *Space Profile Proposal: Partition Manager*.

7.17 Rate Monotonic Manager

Table 15: Rate Monotonic Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_rate_monotonic_cancel()	8	6	6	11
rtems_rate_monotonic_create()	11	4	5	11
rtems_rate_monotonic_delete()	3	3	10	15
rtems_rate_monotonic_get_status()	6	7	7	11
rtems_rate_monotonic_ident()	4	3	12	12
rtems_rate_monotonic_period()	10	5	5	11
rtems_rate_monotonic_reset_all_statistics()	3	5	12	11
rtems_rate_monotonic_reset_statistics()	3	5	12	11
rtems_rate_monotonic_get_statistics()	3	5	12	11

Collected remarks from questionnaire respondents:

- "From CPU load evaluation"
- "Only if timing properties are all known and controllable such that response time analysis can be engaged upon this mechanism"
- "Adding the monotonic statistics (currently not in RTEMS EDISOFT if I am not mistaken), is considered useful for confidence test, V&V activities and some continuous monitoring"
- "Statistics would be nice, but not in the current API state (i.e. over serial)"

See also *Space Profile Proposal: Rate Monotonic Manager*.

7.18 Region Manager

Table 16: Region Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_region_create()	1	2	1	27
rtems_region_delete()	0	2	1	28
rtems_region_extend()	0	1	1	29
rtems_region_get_free_information()	0	1	3	27
rtems_region_get_information()	0	1	2	28
rtems_region_get_segment()	1	1	2	27
rtems_region_get_segment_size()	0	1	3	27
rtems_region_ident()	1	1	2	27
rtems_region_resize_segment()	0	1	1	29
rtems_region_return_segment()	0	1	2	28

Collected remarks from questionnaire respondents:

- "Although not currently using it, it may be a good alternative to an in-house memory manager"
- "Single region to get segments to allocate partitions"

7.19 Semaphore Manager

Table 17: Semaphore Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_semaphore_create()	26	3	0	2
rtems_semaphore_delete()	13	3	6	9
rtems_semaphore_flush()	14	9	3	5
rtems_semaphore_ident()	13	8	5	5
rtems_semaphore_obtain()	25	4	0	2
rtems_semaphore_release()	25	4	0	2
rtems_semaphore_set_priority()	12	9	4	6

Collected remarks from questionnaire respondents:

- "Semaphores are used for tasks synchronisation in our baseline. Only created, obtained and released, but not destroyed. Fixed priorities."

See also *Space Profile Proposal: Semaphore Manager*.

7.20 Signal Manager

Table 18: Signal Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_signal_catch()	5	4	4	18
rtems_signal_send()	5	4	4	18

Collected remarks from questionnaire respondents:

- "Useful for some cases of inter-task communications"
- "Mission codebases are using this API."

7.21 SMP Support

Table 19: SMP Support

Feature	MUST	SHOULD	COULD	WON'T
rtems_get_current_processor()	14	9	4	4
rtems_get_processor_count()	12	8	5	6

7.22 Extensions Manager

Table 20: Extensions Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_extension_create()	9	5	3	14
rtems_extension_delete()	3	3	4	21
rtems_extension_ident()	6	2	7	16

Collected remarks from questionnaire respondents:

- "Just a provision, may be used to check task death"

- "To define extensions for collection of time and stack usage statistics. Extensions created but never destroyed "
- "Only the callback in case of fatal error detection of interest"

See also *Space Profile Proposal: Extensions Manager*.

7.23 Task Manager

Table 21: Task Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_task_create()	29	0	2	0
rtems_task_delete()	13	7	4	7
rtems_task_ident()	18	7	4	2
rtems_task_is_suspended()	13	7	6	5
rtems_task_priority()	18	4	5	4
rtems_task_restart()	11	7	9	4
rtems_task_resume()	14	6	7	4
rtems_task_self()	20	4	6	1
rtems_task_set_affinity()	17	4	6	4
rtems_task_set_priority()	20	6	5	0
rtems_task_set_scheduler()	13	3	12	3
rtems_task_start()	29	0	2	0
rtems_task_suspend()	15	5	8	3
rtems_task_wake_after()	26	1	2	2

Collected remarks from questionnaire respondents:

- "To handle core affinity for tasks running on multi-core processor. Tasks are started but never destroyed or suspended. Fixed waiting periods used for task synchronisation with HW responses"

See also *Space Profile Proposal: Task Manager*.

7.24 Timer Manager

Table 22: Timer Manager

Feature	MUST	SHOULD	COULD	WON'T
rtems_timer_cancel()	15	6	5	5
rtems_timer_create()	19	7	1	4
rtems_timer_fire_after()	17	8	1	5
rtems_timer_ident()	10	8	7	6
rtems_timer_reset()	16	8	3	4

See also *Space Profile Proposal: Timer Manager*.

7.25 Fatal Error Handling

Table 23: Fatal Error Handling

Feature	MUST	SHOULD	COULD	WON'T
rtems_fatal_error_occurred()	12	7	7	5
rtems_fatal()	11	6	9	5
rtems_exception_frame_print()	5	7	10	9
rtems_fatal_source_text()	5	6	11	9
rtems_internal_error_text()	5	6	12	8

Collected remarks from questionnaire respondents:

- "Traps management strategy is based on collecting information and restarting the SW"
- "FDIR is usually done in a mission-specific, custom way."

See also *Space Profile Proposal: Fatal Error Manager*.

7.26 POSIX Barriers

Table 24: POSIX Barriers

Feature	MUST	SHOULD	COULD	WON'T
pthread_barrierattr_destroy()	2	1	1	27
pthread_barrierattr_getpshared()	2	1	2	26
pthread_barrierattr_init()	3	1	1	26
pthread_barrierattr_setpshared()	2	1	2	26
pthread_barrier_destroy()	3	0	1	27
pthread_barrier_init()	4	0	1	26
pthread_barrier_wait()	4	0	1	26

7.27 POSIX Clocks

Table 25: POSIX Clocks

Feature	MUST	SHOULD	COULD	WON'T
clock_getcpuclockid()	1	4	2	24
clock_getres()	2	3	3	23
clock_gettime()	2	6	0	23
clock_nanosleep()	1	4	3	23
clock_settime()	1	5	2	23
nanosleep()	3	4	1	23

7.28 POSIX Condition Variables

Table 26: POSIX Condition Variables

Feature	MUST	SHOULD	COULD	WON'T
pthread_condattr_destroy()	1	3	2	25
pthread_condattr_getclock()	1	3	3	24
pthread_condattr_getshared()	1	2	4	24
pthread_condattr_init()	3	3	1	24
pthread_condattr_setclock()	1	3	3	24
pthread_condattr_setpshared()	1	2	4	24
pthread_cond_broadcast()	1	3	3	24
pthread_cond_destroy()	0	3	2	26
pthread_cond_init()	3	4	0	24
pthread_cond_signal()	3	4	0	24
pthread_cond_timedwait()	3	4	0	24
pthread_cond_wait()	4	3	0	24
pthread_once()	1	3	11	16

Collected remarks from questionnaire respondents:

- "Usage of POSIX not required in our baseline"
- "so far we haven't looked at the POSIX interface (so all related is answered with no) but if this interface is beneficial we might consider using it."

7.29 POSIX Keys

Table 27: POSIX Keys

Feature	MUST	SHOULD	COULD	WON'T
pthread_key_create()	1	2	1	27
pthread_key_delete()	0	0	1	30
pthread_getspecific()	1	2	1	27
pthread_setspecific()	1	2	1	27

7.30 POSIX Message Queues

Table 28: POSIX Message Queues

Feature	MUST	SHOULD	COULD	WON'T
mq_close()	3	0	1	27
mq_getattr()	3	0	1	27
mq_notify()	3	0	1	27
mq_open()	3	0	1	27
mq_receive()	3	0	1	27
mq_send()	3	0	1	27
mq_setattr()	3	0	1	27
mq_timedreceive()	3	0	1	27
mq_timedsend()	3	0	1	27
mq_unlink()	2	0	1	28

7.31 POSIX Mutexes

Table 29: POSIX Mutexes

Feature	MUST	SHOULD	COULD	WON'T
pthread_mutexattr_destroy()	2	1	3	25
pthread_mutexattr_getprioceiling()	3	1	4	23
pthread_mutexattr_getprotocol()	3	1	4	23
pthread_mutexattr_getpshared()	3	0	5	23
pthread_mutexattr_getrobust()	3	1	4	23
pthread_mutexattr_gettype()	3	1	4	23
pthread_mutexattr_init()	4	1	3	23
pthread_mutexattr_setprioceiling()	3	1	4	23
pthread_mutexattr_setprotocol()	3	1	4	23
pthread_mutexattr_setpshared()	3	1	4	23
pthread_mutexattr_setrobust()	3	1	4	23
pthread_mutexattr_settype()	3	1	4	23
pthread_mutex_consistent()	3	2	3	23
pthread_mutex_destroy()	2	2	2	25
pthread_mutex_getprioceiling()	3	2	3	23
pthread_mutex_init()	5	2	1	23
pthread_mutex_lock()	5	2	1	23
pthread_mutex_setprioceiling()	4	2	2	23
pthread_mutex_timedlock()	5	2	1	23
pthread_mutex_trylock()	5	2	1	23
pthread_mutex_unlock()	5	2	1	23

7.32 POSIX Read-Write Locks

Table 30: POSIX Read-Write Locks

Feature	MUST	SHOULD	COULD	WON'T
pthread_rwlockattr_destroy()	1	2	3	25
pthread_rwlockattr_getpshared()	2	1	4	24
pthread_rwlockattr_init()	2	2	3	24
pthread_rwlockattr_setpshared()	2	1	4	24
pthread_rwlock_destroy()	1	2	3	25
pthread_rwlock_init	4	2	1	24
pthread_rwlock_rdlock()	4	2	1	24
pthread_rwlock_timedrdlock()	4	2	1	24
pthread_rwlock_timedwrlock()	4	2	1	24
pthread_rwlock_tryrdlock()	4	2	1	24
pthread_rwlock_trywrlock()	4	2	1	24
pthread_rwlock_unlock()	4	2	1	24
pthread_rwlock_wrlock()	4	2	1	24

7.33 POSIX Semaphores

Table 31: POSIX Semaphores

Feature	MUST	SHOULD	COULD	WON'T
sem_destroy()	2	1	3	25
sem_getvalue()	5	1	2	23
sem_init()	5	1	2	23
sem_post()	5	1	2	23
sem_timedwait()	5	1	2	23
sem_trywait()	5	1	2	23
sem_wait()	5	1	2	23

7.34 POSIX Named Semaphore

Table 32: POSIX Named Semaphore

Feature	MUST	SHOULD	COULD	WON'T
sem_close()	0	0	3	28
sem_open()	0	0	3	28
sem_unlink()	0	0	3	28

7.35 POSIX Spinlocks

Table 33: POSIX Spinlocks

Feature	MUST	SHOULD	COULD	WON'T
pthread_spin_destroy()	1	0	1	29
pthread_spin_init()	2	1	1	27
pthread_spin_lock()	2	1	1	27
pthread_spin_trylock()	2	1	1	27
pthread_spin_unlock()	2	1	1	27

7.36 POSIX Thread Cancellation

Table 34: POSIX Thread Cancellation

Feature	MUST	SHOULD	COULD	WON'T
pthread_cancel()	1	0	2	28
pthread_cleanup_pop()	0	1	2	28
pthread_cleanup_push()	0	1	2	28
pthread_setcancelstate()	0	1	2	28
pthread_testcancel()	0	1	2	28

7.37 POSIX Threads

Table 35: POSIX Threads

Feature	MUST	SHOULD	COULD	WON'T
pthread_attr_destroy()	2	1	2	26
pthread_attr_getdetachstate()	2	2	2	25
pthread_attr_getguardsize()	2	1	3	25
pthread_attr_getinheritsched()	2	2	2	25
pthread_attr_getschedparam()	3	1	2	25
pthread_attr_getschedpolicy()	3	1	2	25
pthread_attr_getscope()	2	2	2	25
pthread_attr_getstack()	2	2	2	25
pthread_attr_getstacksize()	2	2	2	25
pthread_attr_init()	3	2	2	24
pthread_attr_setdetachstate()	2	2	3	24
pthread_attr_setguardsize()	2	1	4	24
pthread_attr_setinheritsched()	2	2	3	24
pthread_attr_setschedparam()	3	1	3	24
pthread_attr_setschedpolicy()	3	1	3	24
pthread_attr_setscope()	2	1	4	24
pthread_attr_setstack()	2	1	4	24
pthread_attr_setstacksize()	3	1	3	24
pthread_create()	4	1	2	24
pthread_detach()	4	0	3	24
pthread_equal()	3	1	3	24
pthread_exit()	2	1	3	25
pthread_getconcurrency()	2	1	4	24
pthread_getcpuclockid()	2	2	3	24
pthread_getname_np()	2	1	3	25
pthread_getschedparam()	3	1	3	24
pthread_join()	3	1	3	24
pthread_self()	3	2	2	24
pthread_setconcurrency()	2	1	4	24
pthread_setname_np()	2	1	3	25
pthread_setschedparam()	3	1	3	24
pthread_setschedprio()	2	2	3	24

Collected remarks from questionnaire respondents:

- "In principle a single thread interface would be enough, either RTEMS or POSIX. I have set the POSIX items in the lower range SHOULD instead of MUST. I have tried to mark which functionality of POSIX I believe is important."

BIBLIOGRAPHY

[CGB14] Felipe Cerqueira, Arpan Gujarati, and Björn B. Brandenburg. Linux's Processor Affinity API, Refined: Shifting Real-Time Tasks towards Higher Schedulability. In *Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS 2014)*. 2014. URL: http://www.mpi-sws.org/~bbb/papers/pdf/rtss14f.pdf.

[ECS09] ECSS. *ECSS-E-ST-40C - Software general requirements*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/.

[ECS17] ECSS. *ECSS-Q-ST-80C (Rev.1) - Software product assurance*. European Cooperation for Space Standardization, 2017. URL: https://ecss.nl/standard/ecss-q-st-80c-rev-1-software-product-assurance-15-february-2017/.

[GCB13] Arpan Gujarati, Felipe Cerqueira, and Björn B. Brandenburg. Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*. 2013. URL: https://people.mpi-sws.org/~bbb/papers/pdf/ecrts13a-rev1.pdf.

