CRITICAL SOFTWARE, SA
EN1/IC2, KM 185,6
BANHOS SECOS, SANTA CLARA
3040-032 COIMBRA, PORTUGAL
TEL. +351.239.801300, FAX. +351.239.801319
WWW.CRITICALSOFTWARE.COM

CRITICAL SOFTWARE, LIMITED
111 NORTH MARKET STREET, SUITE 670
SAN JOSE, CALIFORNIA, USA, 95113
TEL. +1(408) 9711231,
FAX: +1(408) 3513330

RTEMS 4.5.0 Stress Testing Report

RAMS Call-of Order 2

Contract Ref.: *CSW-RAMS-2003-CTR-1306*

ESTEC/Contract Nº 16582/02/NL/PA

Date: *25-11-2003*
Pages: *17*
State: *Approved*
Access: *See Access List*
Reference: *DL-RAMS02-04-02*
CSW-RAMS-2003-RPT-1338-02

Partners / Clients:

RTEMS 4.5.0 Stress Testing Report

RAMS Call-of Order 2

Approved Version: 1.17			
Name	**Function**	**Signature**	**Date**
Ricardo Maia	Project Manager		25-11-2003
José Silva	SQA Engineer		25-11-2003

Authors and Contributors:			
Name	**Contact**	**Description**	**Date**
Ricardo Barbosa	rbarbosa@criticalsoftware.com	Project Engineer	20/10/2003
Ricardo Maia	rmaia@criticalsoftare.com	Project Manager	25/11/2003
Lubomir Velkov	lvelkov@criticalsoftware.com	Senior Engineer	24/7/2003
Luís Henriques	lhenriques@criticalsoftware.com	Senior Engineer	20/10/2003

Access List:
Internal Access
Project Team Members
External Access
ESA-ESTEC

Revision History:			
Version	**Date**	**Description**	**Author**
0.1	31/07/2003	First Draft	Ricardo Barbosa
1.0	20/10/2003	Added test methodology and stress model	Luís Henriques
1.1	28/10/2003	Revision	R. Maia
2.0	30/10/2003	Changed according to internal review	Luís Henriques
2.1	25/11/2003	Updated disclaimer.	R. Maia

Table of Contents

List of Tables, Figures and Equations

1. Introduction

1.1 Objective

This document presents the results of the stress testing performed in the Real Time Executive for Multiprocessor Systems (RTEMS) version 4.5.0. This evaluation is performed in the scope of the Call-off Order number 02 under project Software Dependability and Safety Evaluations, ESTEC/Contract Nº 16582/02/NL/PA.

The main goal of this document is to define the test cases to perform on RTEMS, as well as the result driven from these tests along with the methodology applied in the definition and execution of the tests.

1.2 Scope

This report is the deliverable DL-RAMS02-04 issue 2 of the Call-off Order number 02 under project Software Dependability and Safety Evaluations, ESTEC/Contract Nº 16582/02/NL/PA and presents the results of work packages WP-RAMS02-410 and WP-RAMS02-420.

1.3 Audience

This document targets several groups of readers, namely:

- "Software Dependability and Safety Evaluations" team members and in particular the Call-off Order 2 team members.
- Space software staff involved in the development of on-board software.
- Staff involved in the development of RTEMS related software.
- Space software product assurance staff.
- Management and technical ESA/ESTEC staff.

1.4 Acronyms

Acronyms	Description
API	Application Programming Interface
CPU	Central Processing Unit
CSW	Critical Software, S.A.
RAM	Radom-Access Memory
RAMS	Reliability, Availability, Maintainability and Safety
RTEMS	Real Time Executive for Multiprocessor Systems

Table 1. Acronyms Table

1.5 Document Structure

This document has the following structure:

Chapter 1	Introduces the document, as well as the document scope, intended audience and a list of acronyms and references used through out the document.
Chapter 2	Presents the test methodology used in the stress testing of RTEMS.

Chapter 3	Presents the templates used in this document for test campaigns, test suites and test cases results.
Chapter 4	Presents the results summary achieved in the stress testing.
Annex A	Presents the test campaigns and test suites definition, as well as the results of the tests execution. It also provides an analysis of the results for each of the test cases.
Annex B	Contains the source code for all the workloads used in the stress testing of the RTEMS 4.5.0.

.

Annex A and Annex B are available as a separate volume of this document.

1.6 References

[1] RTEMS 4.5.0 Evaluation Report, DL-RAMS02-01-02, CSW-RAMS-2003-RPT-1334, Ricardo Barbosa, Ricardo Maia, Luís Henriques, Lubomir Velkov, João Esteves, 17/09/2003

[2] RTEMS 4.5.0 Stress Testing Report – Annex, DL-RAMS02-01-02, CSW-RAMS-2003-RPT-1338, Ricardo Barbosa, Ricardo Maia, Luís Henriques, Lubomir Velkov, João Esteves, 28/10/2003

[3] OAR, RTEMS C User's Guide, September 2000

[4] Xception web Site, http://www.xception.org

[5] ECSS-Q-80-03A – "Guidelines for methods and techniques to support the verification and validation of software dependability and safety", 25 July 2003.

2. Test Methodology and Stress Model

Stress testing is a kind of black box testing that burdens the test object with an exceptionally high workload in order to get knowledge on the real limits of the system.

Stress testing might be used to evaluate characteristics such as the time behaviour of the test object, the influence of load, the correct dimension of internal buffers or dynamic variables, stacks, etc.

There are a variety of test conditions which can be applied for stress testing. Some of these conditions are [5]:

- if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;
- if working on demand then the number of demands per time unit to the test object is increased beyond normal conditions (an example of this might be the generation of interrupts at a high rate);
- if the size of a database plays an important role then it is increased beyond normal conditions;
- influential devices are tuned to their maximum speed or lowest speed;
- for the extreme cases, all influential factors are set to the boundary conditions at the same time, as far as is possible.

The usage of resources such as CPU time, storage space and memory are usually subject to stress testing.

2.1 Test Methodology

The approach to be used in the stress testing of the RTEMS 4.5.0 consists in the execution of several workloads that makes extremely high usage of system resources. Two types of resources will be evaluated:

- Physical resources – CPU time and memory;
- Logical resources – task, semaphores, message queues, etc.

Stressing of these resources will be accomplished by:

- Creating a large number of resources (only for logical resources);
- Making intensive use of the created resources using different entities (e.g. a large number of tasks accessing the same message queue).

The usage of the resources will gradually increase until it gets to a value which the system could not stand. This is possible because RTEMS 4.5.0 has no hard-coded limits to the several types of objects that can be created (task, message queues, etc), being this limite defined by the applications at compile time and by the available physical resources (mainly, the memory).

The methodology to use consists of three phases:

- Preparation - including all the activities needed to define the test cases.
- Test Execution - execution of the defined test cases.

- Log Analysis - analysis of the results of the test cases execution and identification of the RTEMS 4.5.0 faults.

2.1.1 Preparation

The Preparation phase comprises four tasks:

a. Product Analysis and Scope Definition: Analysis of the product under evaluation (i.e. the RTEMS 4.5.0) and selection of system resources that will be subjected to stress testing evaluation.

b. Construction of workloads: definition and implementation of applications that will exercise the system resources.

c. Stress Model Definition: Definition of the desired level of usage for the resources under evaluation (e.g. define the number of tasks to be created).

d. Definition of Test Campaigns and Test Suites: definition of the test campaigns, test suites and test cases. Test cases are logically grouped in test suites which in turn are logically grouped in test campaigns.

2.1.2 Test Execution

During this phase the test cases are executed with support of Xception [4]. The results are collected into a database to be analysed in the following phase. This task is performed in a fully automated mode by Xception.

2.1.3 Log Analysis

The final phase of this methodology is the Log Analysis. In this phase a detailed analysis of the log of each test case is performed and the results are reported. Typically a list of product faults is compiled at this time. For this phase it is important to have clear and well structured log outputs from tests, so that the analysis is performed quickly and correctly.

2.2 Product Analysis and Scope Definition

The RTEMS is a real time executive that provides a high performance environment for embedded critical and military applications including the following features:

- Multitasking capabilities;
- Homogeneous and heterogeneous multiprocessor systems support;
- Event-driven, priority based, pre-emptive scheduling;
- Optional rate monotonic scheduling;
- Intertask communication and synchronisation;
- Priority Inheritance mechanisms;
- Responsive interrupt management;
- Dynamic memory allocation;
- High level of user configurability.

The internal architecture for RTEMS can be viewed as a set of layers that work closely with each other in order to provide a set of services to the real time applications. The executive interface presented to the application is formed by directives (RTEMS API Calls) grouped into logical sets called resource managers.

RTEMS 4.5.0 provides several APIs for real time application programming. Only the classic API was subject to this evaluation (see Figure 1). The Classic is the native and older RTEMS API.

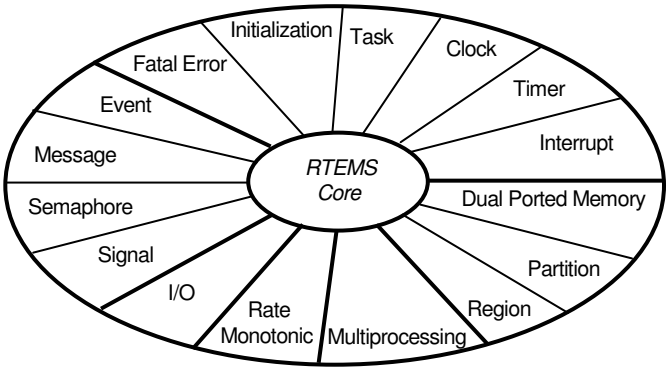

Figure 1. RTEMS Classic API Internal Architecture

The following resources were subjected to stress testing:

- Physical resources:
  - Memory
  - CPU Time
- Logical resources:
  - Tasks
  - Interrupt
  - Partition
  - Signal
  - Semaphores
  - Messages Queues
  - Events

2.3 Workloads

In order to perform the evaluation some applications that exercise the RTEMS directives under test are required. In this evaluation, a set of such applications, called workloads, are used.

Each selected RTEMS resource manager will be stress tested with a different workload. However, every workload follows a similar approach: they all implement a scalable producer/consumer algorithm that uses the selected manager resources.
There are several parameters that can be modified in the workload in order to adjust the degree of load of an RTEMS resource manager. Although there are several workload specific parameters (e.g., the maximum size of a message in a message queue for the

Message Manager workload), some generic parameters are common to all the workloads. These generic parameters are listed in Table 2.

Data Type	Parameter Name	Description
rtems_unsigned32	NUMBER_OF_PRODUCERS	Number of producer tasks
rtems_unsigned32	PRODUCERS_TASK_STACK_SIZE	Producers tasks stack size
rtems_task_priority	PRODUCERS_PRIORITY	Producers priority
rtems_mode	PRODUCERS_TASK_MODE	Producers tasks mode
rtems_attribute	PRODUCERS_TASK_ATTR	Producers tasks attributes
rtems_unsigned32	NUMBER_OF_CONSUMERS	Number of consumers tasks
rtems_unsigned32	CONSUMERS_TASK_STACK_SIZE	Consumers tasks stack size
rtems_mode	CONSUMERS_TASK_MODE	Consumers tasks mode
rtems_attribute	CONSUMERS_TASK_ATTR	Consumers tasks attributes
rtems_task_priority	CONSUMERS_PRIORITY	Consumers priority
rtems_unsigned32	NUMBER_OF_SYSTEMS	Number of systems (producers/consumers systems)
rtems_interval	TEST_TIMEOUT	Number of ticks to wait until workload finishes

Table 2 - Workload Generic Parameters

A very important generic parameter is NUMBER_OF_SYSTEMS. This parameter controls the scalability of the workload. If, for instance, we have only one producer (NUMBER_OF_PRODUCERS with value 1) and two consumers (NUMBER_OF_CONSUMERS with value 2), but the number of systems is set to ten, this means that ten (1 x 10) producers will produce (workload specific) items that will be consumed by twenty (2 x 10) consumers.
All the parameters are defined at workload compilation time through the C pre-processor *#define* directives in the workload header files.

The source code of the defined workloads can be found in Annex B.

2.4 Stress Model

A systematic approach was defined to stress the RTEMS resource managers through the definition of set of values to the different workload parameters. These sets of values were defined in the entire range of possible values for each parameter, taking into account their data type.
When more than one parameter is being modified at the same time within a workload, a combination of all the workload parameters values will be used, obtaining a representative sampling of the parameters interconnection.
To generate the values of the workload parameters, the following formula was used:

$$test\_values = \left\{ 2^{\left(m^{\left(\frac{i}{n+1} \right)} \right)}, \forall i \in \{1, n\} \right\}$$

Equation 1 - Test Values Formula

where *m* is the size of the data type (e.g., 32 for the *rtems_unsigned32* type) and *n* is the desired number of values.

The main goal of this equation was to generate non-linear distribution of test values, in a way that most of the values are small values and only a small percentage of them are large values. The reason for using this type of distribution is that most of the larger values will result in an invalid parameter. For instance, the number of producers tasks is given by an

unsigned 32 bits data type. However, with the available resources on the target system (mainly, the RAM memory), it is not possible to create 2^{32} tasks along with the consumers and all the other resources needed by a workload. With this equation, we reduce the number of invalid parameters that would have no meaning in stress tests.

As all the workload parameters are unsigned 32 bits data types, the *m* was defined with 32. The *n* variable was set to 4, i.e., 4 test values were defined for each workload parameter. The reason for setting a low value to *n* was that the number of test cases explodes with the increase of this variable. For instance, if a test suite mutates 3 workload parameters, with 4 test values, 64 test cases would be generated; if the number of test values was 5, then 125 test cases would be generated.The following table presents the test values to the workload parameters as provided by Equation 1.

n	Test values for 32 bits types
1	4
2	16
3	256
4	65536

Table 3 - Parameters Test Values

2.5 Definition of Test Campaigns and Test Suites

During the preparation phase of this evaluation, test campaigns and test suites were defined. One campaign was defined for each RTEMS resource manager under test. Each campaign includes one test suite for each set of workload parameters that will be mutated.

A total of 7 campaigns were defined, one for each manager.

3. Documentation Templates

This section presents the templates used through out the document in the definition of test campaigns, test suites and test results.

3.1 Test Campaign Definition Template

The following table is the campaign definition template:

Test Campaign Definition	
Campaign Identifier:	
Purpose:	
Workload Files:	
Test Suites:	**1.**
Workload Description:	

Table 4.Test Campaign Definition Template

3.1.1 Campaign Identifier

This field identifies the test campaign name. The following naming convention is used:

RTEMS-CMP-<API INTERFACE ID>-<RTEMS MANAGER ID>

(e.g. RTEMS-CMP-CL-TSK)

- RTEMS – The acronym of the product under evaluation
- CMP – stands for campaign
- <API INTERFACE ID> - Identifier of the API to be tested by the campaign. As only the Classic API is being tested, this identifier will always be *CL*.
- <RTEMS MANAGER ID> - Identifier of the resource manager to be tested by the campaign (refer to Table 5).

RTEMS Manager Abbreviation	Description
TSK	Task Manager
INT	Interrupt Manager
SMP	Semaphore Manager
MSG	Message Manager
EVT	Event Manager
SGL	Signal Manager
PRT	Partition Manager

Table 5 - RTEMS Manager Identifiers

3.1.2 Purpose

This field describes the purpose of the test campaign;

3.1.3 Workload Files

This field provides the name of the related workload files. The workloads source code can be found in Annex B [2].

3.1.4 Test Suites

Lists the test suites of a specific test campaign.

3.1.5 Workload Description

Provides details regarding the workload implementation.

3.2 Test Suite Definition Template

The following table is used in the definition of a test suite.

Test Suite Definition
Test Suite Identifier:
Purpose:
Fault Location(s):
Generated Test Cases:

Table 6. Test Suite Definition Template

3.2.1 Test Suite Identifier

Uniquely identifies the test suite.

The identifier used follows the naming convention:

RTEMS-TS-<API INTERFACE ID>-<RTEMS MANAGER ID>-<SEQUENTIAL NUMBER>

(e.g. RTEMS-TS-CL-TSK-001)

- RTEMS – The acronym of the product under evaluation
- TS – stands for Test Suite
- <API INTERFACE ID> - Identifier of the API tested by the campaign. As only the Classic API is being tested, this identifier will always be *CL*.
- <RTEMS MANAGER ID> - Identifier of the resource manager tested by the campaign (refer to Table 5).
- <SEQUENTIAL NUMBER> - A sequential number, different for each test suite within the same campaign.

3.2.2 Purpose

Describes the purpose of the test suite.

3.2.3 Fault Location(s)

Identifies the workload parameter to modify in the test cases. It includes the filename of the workload (typically, the workload header file) and the line(s) number(s) in which the parameter is defined.

3.2.4 Generated Test Cases

The number of test cases that were generated for the test suite.

3.3 Test Case Result Template

Each of the test cases in which a fault on the RTEMS is uncovered will be presented in this document using the template described in this section.

Test Case Results	
Test case result identifier:	
Input Specification:	
Fault Description:	
Notes:	

Table 7 - Test Case Result Template

3.3.1 Test Case Result Identifier

Uniquely identifies a test case result.

The identifier used follows the naming convention:

RTEMS-TCR-<API INTERFACE ID>-<RTEMS MANAGER ID>-<TEST SUITE NUMBER>-<SEQUENTIAL NUMBER>

(e.g. RTEMS-TCR-CL-TSK-001-001)

- RTEMS – The acronym of the product under evaluation
- TCR – stands for Test Case Result
- <API INTERFACE ID> - Identifier of the API tested by the campaign. As only the Classic API is being tested, this identifier will always be *CL*.
- <RTEMS MANAGER ID> - Identifier of the resource manager tested by the campaign (refer to Table 5).
- <TEST SUITE NUMBER> - The sequential number from the test suite.
- <SEQUENTIAL NUMBER> - A sequential number, different for each test case result within the same test suit.

3.3.2 Input Specification

Specifies the parameter changed and the test value used in the test case.

3.3.3 Fault Description

Describes the fault uncovered in the execution of the test case.

3.3.4 Notes

Provides further information regarding the test case.

4. Results Summary

During the stress testing of the RTEMS 4.5.0 Classic API, 452 test cases were defined. The execution of these test cases resulted in 74 *pass* and 378 *fails*. Table 8 shows the distribution of the test cases and faults among the several RTEMS managers.

RTEMS Manager	Number of Test Cases	Number of Test Cases fails
Task Manager	20	16
Semaphore Manager	68	51
Message Manager	80	74
Signal Manager	68	56
Interrupt Manager	20	14
Event Manager	68	56
Partition Manager	128	111
Total	**452**	**378**

Table 8 - Number of Passed/Failed Test Cases

During the test cases execution, three types of problems were identified. These types of problems occured in different moments of the test case execution. These are:

- **Application linkage:** when the linker fails to create a binary image to execute. The *ld* command issues an error message saying that the *region ram* is full for a specified binary section (the .bss section).

- **RTEMS initialisation:** RTEMS fails to initialise. During the test cases execution, RTEMS failed the initialisation in two different ways. In the first one, it failed after detecting that there was not enough *RAM* memory to initialise the application. This failure is detected and the initialisation is aborted. In the second, RTEMS fails after trying to initialise the application accessing an invalid (inexistent) memory address. In this case, the error is not detected.

- **RTEMS objects initialisation:** the application starts running but exits due to lack of memory to create the necessary RTEMS objects. These objects, e.g. message queues or new tasks, are dynamically created by the workload according to the workload parameters.

The following table provides a synthesis of the distribution of these types of problems in the different workloads.

RTEMS Manager	Application Linkage	RTEMS Initialisation	RTEMS Objects Initialisation	Total
Task Manager	-	16	-	16
Semaphore Manager	19	32	-	51
Message Manager	19	37	18	74
Signal Manager	31	25	-	56
Interrupt Manager	3	11	-	14
Event Manager	31	25	-	56
Partition Manager	19	40	52	111
Total	**122**	**186**	**74**	**378**

Table 9 - Test Cases Failure Distribution

As stated above, the RTEMS initialisation failures may be separated in two categories: detected and not detected. Detected failures are those where the RTEMS initialisation

process aborts and exits with an error; not detected failures lead to an invalid memory access. Table 10 presents the distribution of these two kinds of initialisation failures.

RTEMS Manager	Detected	Not Detected	Total
Task Manager	8	8	12
Semaphore Manager	12	20	32
Message Manager	12	25	37
Signal Manager	12	13	25
Interrupt Manager	4	7	11
Event Manager	12	13	25
Partition Manager	12	28	40
Total	**72**	**114**	**182**

Table 10 - RTEMS Initialisation Failures

All the not detected failures were considered to be critical and although the high number of these failures, they seem to be all related with the same problem: an invalid memory access during the RTEMS initialisation.

Another error related with an incorrect error code that is returned by a partition manager directive was found. This fault was detected by 16 of the test cases. It was classified as minor

Table 11 presents the RTEMS faults according to its criticality.

Critical	Minor
1	1

Table 11- RTEMS 4.5.0 Faults by Criticality