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 - Annex

RAMS Call-of Order 02

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

ESTEC/Contract Nº 16582/02/NL/PA

Date: *25-11-2003*
Pages: *157*
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 - Annex

RAMS Call-of Order 02

Approved Version: 1.9			
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	28/10/2003
Ricardo Maia	rmaia@criticalsoftare.com	Project Manager	25/11/2003
Lubomir Velkov	lvelkov@criticalsoftware.com	Senior Engineer	31/7/2003
Luís Henriques	lhenriques@criticalsoftware.com	Senior Engineer	28/10/2003

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

Revision History:			
Version	**Date**	**Description**	**Author**
0.1	31/7/2003	First Draft	Ricardo Barbosa
1.0	14/10/2003	Added the test cases definition	Luís Henriques
1.1	28/10/2003	Added test cases results	Luís Henriques
2.0	30/10/2003	Changed according to document review	Luís Henriques
2.1	25/11/2003	Updated disclaimer	Ricardo Maia

Table of Contents

This document is an annex of the deliverable DL-RAMS02-04-02 of the Call-of-order number 02 under project Software Dependability and Safety Evaluations, ESTEC/Contract Nº 16582/02/NL/PA. It presents the test campaigns and test suites definition as well as the results of the execution and corresponding analysis. It also contains the source code of all of the workloads used during the stress evaluation of the RTEMS 4.5.0.

Annex
A

Annex A. RTEMS Stress Tests Definition and Results

This annex contains the stress tests definition performed in the Real Time Executive for Multiprocessor Systems (RTEMS) Classic API. It also contains the results of the execution and corresponding analysis.

One campaign was defined for each of the RTEMS resource managers under test. Each campaign includes several test suites, which gradually increment the grade of stressing of the corresponding manager. These test suites are composed by several test cases.

All the manager's workloads implement a generic producer/consumer algorithm that stresses the corresponding manager in different ways. Each of the workloads creates a (configurable) number of systems. A system is a set of producers and consumers that share the resources that are subject to stress testing. For instance, a system in the Semaphore Manager workload will contain several producers that will signal a semaphore where several consumers are blocked. The workload can create a configurable number of systems, each system containing a configurable number of producers and a configurable number of consumers.

The manager's workloads define several parameters that shall be manipulated during the test execution in order to increase/decrease the level of stress of the corresponding manager. However, several common parameters are defined in all the managers. These generic parameters are presented in Table 1.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	NUMBER_OF_PRODUCERS	64	Number of producer tasks
rtems_unsigned32	PRODUCERS_TASK_STACK_SIZE	(RTEMS_MINIMUM_STACK_SIZE * 1)	Stack size for the producers tasks
rtems_task_priority	PRODUCERS_PRIORITY	3	Producers priority
rtems_mode	PRODUCERS_TASK_MODE	RTEMS_DEFAULT_MODES	Producers tasks mode
rtems_attribute	PRODUCERS_TASK_ATTR	RTEMS_DEFAULT_ATTRIBUTES	Producers tasks attributes
rtems_unsigned32	NUMBER_OF_CONSUMERS	64	Number of consumers tasks
rtems_unsigned32	CONSUMERS_TASK_STACK_SIZE	(RTEMS_MINIMUM_STACK_SIZE * 1)	Stack size for the consumers tasks
rtems_mode	CONSUMERS_TASK_MODE	RTEMS_DEFAULT_MODES	Consumers tasks mode
rtems_attribute	CONSUMERS_TASK_ATTR	RTEMS_DEFAULT_ATTRIBUTES	Consumers tasks attributes
rtems_task_priority	CONSUMERS_PRIORITY	3	Consumers priority
rtems_unsigned32	NUMBER_OF_SYSTEMS	4	Number of systems (producers/consumers systems)
rtems_interval	TEST_TIMEOUT	20000	Number of ticks to wait until workload finishes

Table 1 - Workload Generic Parameters

These workload parameters (along with the workload specific parameters) are used to set the RTEMS applications configuration. For instance, the RTEMS parameter `CONFIGURE_MAXIMUM_TASKS` is always set to `(NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS + NUMBER_OF_CONSUMERS) + 1)` in the workloads in order to set the maximum number of tasks that the application can create to the needed value for a specific test case.

In order to completely understand the test results and its analysis, the reading of the RTEMS documentation is recommended, specially the RTEMS C User's Guide.

A.1. RTEMS Task Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-TSK
Purpose:	To stress test the task manager.
Workload Files:	rtems-cmp-cl-tsk.c, rtems-cmp-cl-tsk.h, tsk-producer.c, tsk-consumer.c
Test Suites:	1. RTEMS-TS-CL-TSK-001 2. RTEMS-TS-CL-TSK-002
Workload Description: This workload will create a semaphore for each system. The consumers of a specific system will all wait in the system semaphore, while the producers will periodically signal that semaphore to unblock the consumers. The stressing will consist on the modification of the number of threads created (consumers and producers), using two different priorities: an higher priority for the consumers and a lower priority for the producers.	

The following table presents the task manager workload specific parameters with its default values and the generic parameters with its specific values for this workload.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	SEMAPHORES_INITIAL_COUNT	0	Semaphores initial count
rtems_attribute	SEMAPHORES_ATTRIBUTES	RTEMS_DEFAULT_ATTRIBUTES	Semaphores attributes
rtems_task_priority	SEMAPHORES_PRIORITY	RTEMS_NO_PRIORITY	Semaphores priority
rtems_unsigned32	NUMBER_OF_PRODUCED_ITEMS	100	Number of semaphore signals for each producer
rtems_interval	DELAY_BETWEEN_ITEMS	1	Time to wait before producing another item (signaling the semaphore)
rtems_task_priority	PRODUCERS_PRIORITY	3	Producers priority
rtems_task_priority	CONSUMERS_PRIORITY	2	Consumers priority

A.1.1. RTEMS-TS-CL-TSK-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-TSK-001
Purpose:	To stress the task manager by creating as much consumers and producers as possible.

Fault Location(s):	Source file: rtems-cmp-cl.tsk.h
	Lines: [50, 65]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_CONSUMERS 64`

Generated Test Cases:	16

A.1.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-TSK-001-004 (same results obtained in RTEMS-TCR-CL-TSK-001-008, RTEMS-TCR-CL-TSK-001-010, RTEMS-TCR-CL-TSK-001-011, RTEMS-TCR-CL-TSK-001-012)[1]

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-TSK-001-005 (same results obtained in RTEMS-TCR-CL-TSK-001-009, RTEMS-TCR-CL-TSK-001-013, RTEMS-TCR-CL-TSK-001-014, RTEMS-TCR-CL-TSK-001-015, RTEMS-TCR-CL-TSK-001-016, RTEMS-TCR-CL-TSK-001-017)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: After further analysis, the problem was identified and it was an illegal memory access in the RTEMS initialisation code: when trying to initialise the application, the code verifies if there is a remote monitor running. After some calculations using the variables that contain the memory start and end addresses, the memory start address ends up with a value that is too large for the actual system.	

A.1.2. RTEMS-TS-CL-TSK-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-TSK-002
Purpose:	To stress the task manager by creating the default number of tasks with an increasing stack size. Both parameters, the producers and consumers tasks stack sizes, are modified in the same mutation to the same value. The RTEMS `CONFIGURE_EXTRA_TASK_STACKS` parameter needed also to be adjusted, otherwise the workload would not be able to create tasks with a stack size different from the `RTEMS_MINIMUM_STACK_SIZE` constant.

Fault Location(s):	Source file: rtems-cmp-cl.tsk.h
	Lines: [53, 68]
	`#define PRODUCERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)`
	`#define CONSUMERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)`

Generated Test Cases:	4

[1] The referred test cases although having different values for the modified parameters, had very similar outcomes. These test cases are just identified here for statistic reasons and are not described in this document.

A.1.1.2. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-TSK-002-019 (same results obtained in RTEMS-TCR-CL-TSK-001-020, RTEMS-TCR-CL-TSK-001-022)

Input Specification:
`#define PRODUCERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 4)`
`#define CONSUMERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 4)`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

A.1.1.3. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-TSK-001-021

Input Specification:
`#define PRODUCERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 256)`
`#define CONSUMERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 256)`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

A.2. RTEMS Semaphore Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-SMP
Purpose:	To stress test the semaphore manager.
Workload Files:	rtems-cmp-cl-smp.c, rtems-cmp-cl-smp.h, smp-producer.c, smp-consumer.c
Test Suites:	1. RTEMS-TS-CL-SMP-001 2. RTEMS-TS-CL-SMP-002
Workload Description:	This workload will create a semaphore for each system. The consumers of a specific system will all wait in the system semaphore, while the producers will periodically signal that semaphore to unblock the consumers. The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that access the semaphores within each system.

The following table presents the task manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	SEMAPHORES_INITIAL_COUNT	0	Semaphores initial count
rtems_attribute	SEMAPHORES_ATTRIBUTES	RTEMS_DEFAULT_ATTRIBUTES	Semaphores attributes
rtems_task_priority	SEMAPHORES_PRIORITY	RTEMS_NO_PRIORITY	Semaphores priority
rtems_unsigned32	NUMBER_OF_PRODUCED_ITEMS	1000	Number of semaphore signals for each producer
rtems_interval	DELAY_BETWEEN_ITEMS	1	Time to wait before producing another item (signaling the semaphore)
rtems_task_priority	PRODUCERS_PRIORITY	3	Producers priority
rtems_task_priority	CONSUMERS_PRIORITY	2	Consumers priority

A.2.1. RTEMS-TS-CL-SMP-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-SMP-001
Purpose:	To stress the semaphore manager by increasing the number of systems (which will result in the creation of more semaphores) and the number of tasks that use the semaphore (consumers and producers).

Fault Location(s):	Source file: rtems-cmp-cl-smp.h
	Lines: [50, 65, 80]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_CONSUMERS 64`
	`#define NUMBER_OF_SYSTEMS 4`

Generated Test Cases:	64

A.2.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SMP-001-004 (same results obtained in RTEMS-TCR-CL-SMP-001-008, RTEMS-TCR-CL-SMP-001-010, RTEMS-TCR-CL-SMP-001-011, RTEMS-TCR-CL-SMP-001-012, RTEMS-TCR-CL-SMP-001-020, RTEMS-TCR-CL-SMP-001-024, RTEMS-TCR-CL-SMP-001-026, RTEMS-TCR-CL-SMP-001-027, RTEMS-TCR-CL-SMP-001-034, RTEMS-TCR-CL-SMP-001-035, RTEMS-TCR-CL-SMP-001-038)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SMP-001-005 (same results obtained in RTEMS-TCR-CL-SMP-001-009, RTEMS-TCR-CL-SMP-001-013, RTEMS-TCR-CL-SMP-001-014, RTEMS-TCR-CL-SMP-001-015, RTEMS-TCR-CL-SMP-001-016, RTEMS-TCR-CL-SMP-001-017, RTEMS-TCR-CL-SMP-001-021, RTEMS-TCR-CL-SMP-001-025, RTEMS-TCR-CL-SMP-001-028, RTEMS-TCR-CL-SMP-001-036, RTEMS-TCR-CL-SMP-001-039, RTEMS-TCR-CL-SMP-001-040, RTEMS-TCR-CL-SMP-001-042, RTEMS-TCR-CL-SMP-001-043, RTEMS-TCR-CL-SMP-001-044, RTEMS-TCR-CL-SMP-001-050, RTEMS-TCR-CL-SMP-001-051, RTEMS-TCR-CL-SMP-001-054, RTEMS-TCR-CL-SMP-001-055)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SMP-001-037 (same results obtained in RTEMS-TCR-CL-SMP-001-041, RTEMS-TCR-CL-SMP-001-045, RTEMS-TCR-CL-SMP-001-046, RTEMS-TCR-CL-SMP-001-047, RTEMS-TCR-CL-SMP-001-048, RTEMS-TCR-CL-SMP-001-049, RTEMS-TCR-CL-SMP-001-052, RTEMS-TCR-CL-SMP-001-053, RTEMS-TCR-CL-SMP-001-056, RTEMS-TCR-CL-SMP-001-057, RTEMS-TCR-CL-SMP-001-058, RTEMS-TCR-CL-SMP-001-059, RTEMS-TCR-CL-SMP-001-060, RTEMS-TCR-CL-SMP-001-061, RTEMS-TCR-CL-SMP-001-062, RTEMS-TCR-CL-SMP-001-063, RTEMS-TCR-CL-SMP-001-064, RTEMS-TCR-CL-SMP-001-065)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 256`	

Failure Description: The workload compiled but did not linked, as the linker could not create a binary image that could be loaded into the available amount of RAM memory.	
Notes: The linker error was: `/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-smp.exe section .bss)`	

```
collect2: ld returned 1 exit status
make: *** [o-optimize/rtems-cmp-cl-smp] Error 1
```

A.2.2. RTEMS-TS-CL-SMP-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-SMP-002
Purpose:	To stress the semaphore manager by increasing the number of items produced, i.e., the producers semaphore signaling.
Fault Location(s):	Source file: rtems-cmp-cl-smp.h Lines: [99] `#define NUMBER_OF_PRODUCED_ITEMS 1000`
Generated Test Cases:	4

A.2.2.1. Test Case Results

No faults were detected in this RTEMS task manager with the test cases defined within this test suite.

A.3. RTEMS Message Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-MSG
Purpose:	To stress test the message manager.
Workload Files:	rtems-cmp-cl-msg.c, rtems-cmp-cl-msg.h, msg-producer.c, msg-consumer.c
Test Suites:	1. RTEMS-TS-CL-MSG-001 2. RTEMS-TS-CL-MSG-002

Workload Description:

This workload will create an RTEMS message queue object for each system. The consumers of a specific system will all wait in the system message queue for the arrival of new messages, while the producers will send messages with a certain frequency. The consumers, after receiving a new message, will check the message size and block until another message arrives.

The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that access the message queues within each system. Other parameters that will be used in the message manager stressing is the size of the messages sent to the queues and the frequency used to send a message.

The following table presents the message manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	MAX_MESSAGES	13	Number of messages to be sent by each producer. The message queues will be created with the *count* parameter set to (MAX_MESSAGES * NUMBER_OF_PRODUCERS).
rtems_unsigned32	MAX_MESSAGE_SIZE	32	Messages maximum size
rtems_attribute	MESSAGE_QUEUE_ATTRIBUTES	(RTEMS_FIFO | RTEMS_LOCAL)	Message queues attributes
rtems_interval	DELAY_BETWEEN_ITEMS	1	Time to wait before producing another item (sending another message)
rtems_unsigned32	NUMBER_OF_PRODUCERS	8	Number of producer tasks
rtems_unsigned32	NUMBER_OF_CONSUMERS	8	Number of consumer tasks
rtems_unsigned32	NUMBER_OF_SYSTEMS	2	Number of systems (producers/consumers systems)
rtems_unsigned32	PRODUCERS_PRIORITY	3	Producers priority
rtems_unsigned32	CONSUMERS_PRIORITY	2	Consumers priority

For this specific workload, the values for the `MAX_MESSAGES` and `MAX_MESSAGE_SIZE` parameters needed to be properly selected, as these are very sensitive parameters. Some default values for the generic parameters had also to be modified, namely the number of producers, consumers and systems.

A.3.1. RTEMS-TS-CL-MSG-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-MSG-001
Purpose:	To stress the message manager by creating as many systems as possible. The producer's priority shall be lower than the consumers.

Fault Location(s):	Source file: rtems-cmp-cl-msg.h
	Lines: [50, 65, 80]
	`#define NUMBER_OF_PRODUCERS 8`
	`#define NUMBER_OF_CONSUMERS 8`
	`#define NUMBER_OF_SYSTEMS 2`

Generated Test Cases:	64

A.3.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-001-004 (same results obtained in RTEMS-TCR-CL-MSG-001-008, RTEMS-TCR-CL-MSG-001-010, RTEMS-TCR-CL-MSG-001-011, RTEMS-TCR-CL-MSG-001-012, RTEMS-TCR-CL-MSG-001-020, RTEMS-TCR-CL-MSG-001-024, RTEMS-TCR-CL-MSG-001-026, RTEMS-TCR-CL-MSG-001-027, RTEMS-TCR-CL-MSG-001-034, RTEMS-TCR-CL-MSG-001-035, RTEMS-TCR-CL-MSG-001-038)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-001-005 (same results obtained in RTEMS-TCR-CL-MSG-001-009, RTEMS-TCR-CL-MSG-001-013, RTEMS-TCR-CL-MSG-001-014, RTEMS-TCR-CL-MSG-001-015, RTEMS-TCR-CL-MSG-001-016, RTEMS-TCR-CL-MSG-001-017, RTEMS-TCR-CL-MSG-001-021, RTEMS-TCR-CL-MSG-001-025, RTEMS-TCR-CL-MSG-001-028, RTEMS-TCR-CL-MSG-001-029, RTEMS-TCR-CL-MSG-001-030, RTEMS-TCR-CL-MSG-001-031, RTEMS-TCR-CL-MSG-001-032, RTEMS-TCR-CL-MSG-001-033, RTEMS-TCR-CL-MSG-001-036, RTEMS-TCR-CL-MSG-001-039, RTEMS-TCR-CL-MSG-001-040, RTEMS-TCR-CL-MSG-001-042, RTEMS-TCR-CL-MSG-001-043, RTEMS-TCR-CL-MSG-001-044, RTEMS-TCR-CL-MSG-001-050, RTEMS-TCR-CL-MSG-001-051, RTEMS-TCR-CL-MSG-001-054, RTEMS-TCR-CL-MSG-001-055)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-001-037 (same results obtained in RTEMS-TCR-CL-MSG-001-041, RTEMS-TCR-CL-MSG-001-045, RTEMS-TCR-CL-MSG-001-046, RTEMS-TCR-CL-MSG-001-047, RTEMS-TCR-CL-MSG-001-048, RTEMS-TCR-CL-MSG-001-049, RTEMS-TCR-CL-MSG-001-052, RTEMS-TCR-CL-MSG-001-053, RTEMS-TCR-CL-MSG-001-056, RTEMS-TCR-CL-MSG-001-057, RTEMS-TCR-CL-MSG-001-058, RTEMS-TCR-CL-MSG-001-059, RTEMS-TCR-CL-MSG-001-060, RTEMS-TCR-CL-MSG-001-061, RTEMS-TCR-CL-MSG-001-062, RTEMS-TCR-CL-MSG-001-063, RTEMS-TCR-CL-MSG-001-064, RTEMS-TCR-CL-MSG-001-065)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 256`	

Failure Description: The workload compiled but did not linked, as the linker could not created a binary image that could be loaded into the available amount of RAM memory.	

Notes:
The linker error was:
`/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-msg.exe section .bss)`
`collect2: ld returned 1 exit status`
`make: *** [o-optimize/rtems-cmp-cl-msg] Error 1`	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-001-003 (same results obtained in RTEMS-TCR-CL-MSG-001-007, RTEMS-TCR-CL-MSG-001-018, RTEMS-TCR-CL-MSG-001-019, RTEMS-TCR-CL-MSG-001-022, RTEMS-TCR-CL-MSG-001-023)

Input Specification:
`#define NUMBER_OF_PRODUCERS 16`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: An error code was returned while creating the systems tasks. The error code, `RTEMS_UNSATISFIED`, means that there was no space left in memory for the task stack/FP context.	
Notes:	

A.3.2. RTEMS-TS-CL-MSG-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-MSG-002
Purpose:	To stress the message manager by using changing the number of messages each producer can send and using different messages sizes

Fault Location(s):	Source file: rtems-cmp-cl-msg.h
	Lines: [90, 93]
	`#define MAX_MESSAGES 13`
	`#define MAX_MESSAGE_SIZE 32`

Generated Test Cases:	16

A.3.2.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-002-069 (same results obtained in RTEMS-TCR-CL-MSG-002-070, RTEMS-TCR-CL-MSG-002-073, RTEMS-TCR-CL-MSG-002-074, RTEMS-TCR-CL-MSG-002-077, RTEMS-TCR-CL-MSG-002-078, RTEMS-TCR-CL-MSG-002-079, RTEMS-TCR-CL-MSG-002-080, RTEMS-TCR-CL-MSG-002-081, RTEMS-TCR-CL-MSG-002-082)

Input Specification:
`#define MAX_MESSAGES 256`
`#define MAX_MESSAGE_SIZE 4`	

Failure Description: An error was returned while creating a message queue object. The returned error was `RTEMS_TOO_MANY`, an error that indicates that there are too many global objects.	
Notes: Typically, this error means that the application is trying to create more objects than the value that was set in the RTEMS configuration. However, in the case of the message queues, there is only one RTEMS configuration parameter: the number of message queues to use in the application. The messages size and the number of messages for each message queue are set dynamically at runtime.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-MSG-002-075 (same results obtained in RTEMS-TCR-CL-MSG-001-076)

Input Specification:
`#define MAX_MESSAGES 4`
`#define MAX_MESSAGE_SIZE 256`	

Failure Description: An error was returned while creating a task. The returned error was `RTEMS_UNSATISFIED`, an error that indicates that there is no memory for stack/FP context.	
Notes:	

A.4. RTEMS Signal Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-SGN
Purpose:	To stress test the signal manager.
Workload Files:	rtems-cmp-cl-sgn.c, rtems-cmp-cl-sgn.h, sgn-producer.c, sgn-consumer.c
Test Suites:	1. RTEMS-TS-CL-SGN-001 2. RTEMS-TS-CL-SGN-002
Workload Description:	In this workload, each consumer will register an Asynchronous Signal Routine (ASR) to handle RTEMS signals. After this, the task will enter in an infinite loop, blocking, on each iteration, in a semaphore (each consumer has its own semaphore). Every time a signal is received (sent by a producer), the ASR will update a counter in an array that is indexed by the type of signal received. The producer will just send signals to the producers. Each producer sends every possible type of signal (from RTEMS_SIGNAL_0 to NUMBER_OF_SIGNALS) to each consumer and the signals its semaphore. After sending a signal and signaling the semaphore, the producer releases the processor. The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that send signals and register ASRs within each system. The number of signals will always be a multiple of 32 (the number of defined RTEMS signals). After sending the 32 signals, a producer will restart in the first signal, until it has sent the predefined number of signals.

The following table presents the signal manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_mode	SIGNAL_HANDLER_MODE	RTEMS_PREEMPT	Mode of the Asynchronous Signal Routine (ASR)
rtems_unsigned32	SEMAPHORES_INITIAL_COUNT	0	Semaphores initial count
rtems_attribute	SEMAPHORES_ATTRIBUTES	RTEMS_DEFAULT_ATTRIBUTES	Semaphores attributes
rtems_task_priority	SEMAPHORES_PRIORITY	RTEMS_NO_PRIORITY	Semaphores priority
rtems_unsigned32	NUMBER_OF_SIGNALS	320	Number of signals each producer has to send
rtems_unsigned32	PRODUCERS_PRIORITY	3	Producers priority
rtems_unsigned32	CONSUMERS_PRIORITY	2	Consumers priority

A.4.1. RTEMS-TS-CL-SGN-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-SGN-001
Purpose:	To stress the signal manager by creating as many systems as possible. The producer's priority shall be lower than the consumers.

Fault Location(s):	Source file: rtems-cmp-cl-sgn.h
	Lines: [50, 65, 83]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_CONSUMERS 64`
	`#define NUMBER_OF_SYSTEMS 4`

Generated Test Cases:	64

A.4.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SGN-001-004 (same results obtained in RTEMS-TCR-CL-SGN-001-008, RTEMS-TCR-CL-SGN-001-010, RTEMS-TCR-CL-SGN-001-011, RTEMS-TCR-CL-SGN-001-012, RTEMS-TCR-CL-SGN-001-020, RTEMS-TCR-CL-SGN-001-024, RTEMS-TCR-CL-SGN-001-026, RTEMS-TCR-CL-SGN-001-027, RTEMS-TCR-CL-SGN-001-034, RTEMS-TCR-CL-SGN-001-035, RTEMS-TCR-CL-SGN-001-038)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SGN-001-005 (same results obtained in RTEMS-TCR-CL-SGN-001-009, RTEMS-TCR-CL-SGN-001-013, RTEMS-TCR-CL-SGN-001-021, RTEMS-TCR-CL-SGN-001-025, RTEMS-TCR-CL-SGN-001-028, RTEMS-TCR-CL-SGN-001-029, RTEMS-TCR-CL-SGN-001-036, RTEMS-TCR-CL-SGN-001-037, RTEMS-TCR-CL-MSG-001-038, RTEMS-TCR-CL-MSG-001-042, RTEMS-TCR-CL-MSG-001-043, RTEMS-TCR-CL-MSG-001-044)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-SGN-001-014 (same results obtained in RTEMS-TCR-CL-SGN-001-015, RTEMS-TCR-CL-SGN-001-016, RTEMS-TCR-CL-SGN-001-017, RTEMS-TCR-CL-SGN-001-030, RTEMS-TCR-CL-SGN-001-031, RTEMS-TCR-CL-SGN-001-032, RTEMS-TCR-CL-SGN-001-033, RTEMS-TCR-CL-SGN-001-037, RTEMS-TCR-CL-SGN-001-041, RTEMS-TCR-CL-SGN-001-045, RTEMS-TCR-CL-SGN-001-046, RTEMS-TCR-CL-SGN-001-047, RTEMS-TCR-CL-SGN-001-048, RTEMS-TCR-CL-SGN-001-049, RTEMS-TCR-CL-SGN-001-050, RTEMS-TCR-CL-SGN-001-051, RTEMS-TCR-CL-SGN-001-052, RTEMS-TCR-CL-SGN-001-053, RTEMS-TCR-CL-SGN-001-054, RTEMS-TCR-CL-SGN-001-055, RTEMS-TCR-CL-SGN-001-056, RTEMS-TCR-CL-SGN-001-057, RTEMS-TCR-CL-SGN-001-058, RTEMS-TCR-CL-SGN-001-059, RTEMS-TCR-CL-SGN-001-060, RTEMS-TCR-CL-SGN-001-061, RTEMS-TCR-CL-SGN-001-062, RTEMS-TCR-CL-SGN-001-063, RTEMS-TCR-CL-SGN-001-064, RTEMS-TCR-CL-SGN-001-065)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 256`	

Failure Description: The workload compiled but did not linked, as the linker could not create a binary image that could be loaded into the available amount of RAM memory.	
Notes:	

The linker error was:

```
/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-sgn.exe section .bss)
collect2: ld returned 1 exit status
make: *** [o-optimize/rtems-cmp-cl-sgn] Error 1
```

A.4.2. RTEMS-TS-CL-SGN-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-SGN-002
Purpose:	To stress the signal manager by configuring the producers to send as many signals as possible. The number of producers shall also be incremented with the number of systems.
Fault Location(s):	Source file: rtems-cmp-cl-sgn.h Lines: [105] `#define NUMBER_OF_SIGNALS 320`
Generated Test Cases:	4

A.4.2.1. Test Case Results

No faults were detected in this RTEMS task manager with the test cases defined within this test suite.

A.5. RTEMS Interrupt Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-INT
Purpose:	To stress test the interrupt manager.
Workload Files:	rtems-cmp-cl-int.c, rtems-cmp-cl-int.h, int-producer.c, int-consumer.c
Test Suites:	1. RTEMS-TS-CL-INT-001 2. RTEMS-TS-CL-INT-002
Workload Description: This workload will start by registering two Interrupt Service Routines (ISRs): one for the illegal instruction exception and another to the memory not aligned exception. Then, producer tasks will generate these two kinds of exceptions. When an exception occurs, the trap handler will unlock a consumer by signaling a semaphore, where the consumers are blocked. The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that cause interruptions within each system.	

The following table presents the interrupt manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	NUMBER_OF_INTERRUPTS	100	Number of interrupts that each producer shall generate.

A.5.1. RTEMS-TS-CL-INT-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-INT-001
Purpose:	To stress the interrupt manager by creating as many systems as possible.

Fault Location(s):	Source file: rtems-cmp-cl-int.h
	Lines: [50, 80]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_SYSTEMS 4`

Generated Test Cases:	16

A.5.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-INT-001-004 (same results obtained in RTEMS-TCR-CL-INT-001-006, RTEMS-TCR-CL-INT-001-007, RTEMS-TCR-CL-INT-001-008)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-INT-001-005 (same results obtained in RTEMS-TCR-CL-INT-001-009, RTEMS-TCR-CL-INT-001-010, RTEMS-TCR-CL-INT-001-011, RTEMS-TCR-CL-INT-001-012, RTEMS-TCR-CL-INT-001-014, RTEMS-TCR-CL-INT-001-015)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-INT-001-013 (same results obtained in RTEMS-TCR-CL-INT-001-016, RTEMS-TCR-CL-INT-001-017)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_SYSTEMS 256`	

Failure Description: The workload compiled but did not linked, as the linker could not create a binary image that could be loaded into the available amount of RAM memory.	

Notes:
The linker error was:
`/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-int.exe section .bss)`
`collect2: ld returned 1 exit status`
`make: *** [o-optimize/rtems-cmp-cl-int] Error 1`	

A.5.2. RTEMS-TS-CL-INT-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-INT-002
Purpose:	To stress the interrupt manager by generating large number of interrupts.
Fault Location(s):	Source file: rtems-cmp-cl-int.h Lines: [90] `#define NUMBER_OF_INTERRUPTS 100`
Generated Test Cases:	4

A.5.2.1. Test Case Results

No faults were detected in this RTEMS task manager with the test cases defined within this test suite.

A.6. RTEMS Event Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-EVT
Purpose:	To stress test the event manager.
Workload Files:	rtems-cmp-cl-evt.c, rtems-cmp-cl-evt.h, evt-producer.c, evt-consumer.c
Test Suites:	1. RTEMS-TS-CL-EVT-001 2. RTEMS-TS-CL-EVT-002
Workload Description: In this workload, each consumer will enter in an infinite loop, blocking in each iteration waiting for an event to be delivered. Every time an event is received (sent by a producer), the consumer will update a counter in an array that is indexed by the type of event received. The producers send every possible type of event (from RTEMS_EVENT_0 to NUMBER_OF_EVENTS) to each consumer. After sending an event, the producer releases the processor. The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that send/receive events within each system. The number of events will always be a multiple of 32 (the number of defined RTEMS events). After sending the 32 events, a producer will restart in the first event, until it has sent the predefined number of events.	

The following table presents the event manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	NUMBER_OF_EVENTS	320	Number of RTEMS events that each producer shall send.

A.6.1. RTEMS-TS-CL-EVT-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-EVT-001
Purpose:	To stress the event manager by creating as many systems as possible. The producer's priority shall be lower than the consumers.

Fault Location(s):	Source file: rtems-cmp-cl-evt.h
	Lines: [50, 65, 83]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_CONSUMERS 64`
	`#define NUMBER_OF_SYSTEMS 4`

Generated Test Cases:	64

A.6.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-EVT-001-004 (same results obtained in RTEMS-TCR-CL-EVT-001-008, RTEMS-TCR-CL-INT-001-010, RTEMS-TCR-CL-EVT-001-011, RTEMS-TCR-CL-EVT-001-012, RTEMS-TCR-CL-EVT-001-020, RTEMS-TCR-CL-EVT-001-024, RTEMS-TCR-CL-EVT-001-026, RTEMS-TCR-CL-EVT-001-027, RTEMS-TCR-CL-EVT-001-034, RTEMS-TCR-CL-EVT-001-035, RTEMS-TCR-CL-EVT-001-038)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-EVT-001-005 (same results obtained in RTEMS-TCR-CL-EVT-001-009, RTEMS-TCR-CL-EVT-001-013, RTEMS-TCR-CL-EVT-001-021, RTEMS-TCR-CL-EVT-001-025, RTEMS-TCR-CL-EVT-001-028, RTEMS-TCR-CL-EVT-001-029, RTEMS-TCR-CL-EVT-001-36, RTEMS-TCR-CL-EVT-001-039, RTEMS-TCR-CL-EVT-001-040, RTEMS-TCR-CL-EVT-001-042, RTEMS-TCR-CL-EVT-001-043, RTEMS-TCR-CL-EVT-001-044)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-EVT-001-014 (same results obtained in RTEMS-TCR-CL-EVT-001-015, RTEMS-TCR-CL-EVT-001-016, RTEMS-TCR-CL-EVT-001-017, RTEMS-TCR-CL-EVT-001-030, RTEMS-TCR-CL-EVT-001-031, RTEMS-TCR-CL-EVT-001-032, RTEMS-TCR-CL-EVT-001-033, RTEMS-TCR-CL-EVT-001-037, RTEMS-TCR-CL-EVT-001-041, RTEMS-TCR-CL-EVT-001-045, RTEMS-TCR-CL-EVT-001-046, RTEMS-TCR-CL-EVT-001-047, RTEMS-TCR-CL-EVT-001-048, RTEMS-TCR-CL-EVT-001-049, RTEMS-TCR-CL-EVT-001-050, RTEMS-TCR-CL-EVT-001-051, RTEMS-TCR-CL-EVT-001-052, RTEMS-TCR-CL-EVT-001-053, RTEMS-TCR-CL-EVT-001-054, RTEMS-TCR-CL-EVT-001-055, RTEMS-TCR-CL-EVT-001-056, RTEMS-TCR-CL-EVT-001-057, RTEMS-TCR-CL-EVT-001-058, RTEMS-TCR-CL-EVT-001-059, RTEMS-TCR-CL-EVT-001-060, RTEMS-TCR-CL-EVT-001-061, RTEMS-TCR-CL-EVT-001-062, RTEMS-TCR-CL-EVT-001-063, RTEMS-TCR-CL-EVT-001-064, RTEMS-TCR-CL-EVT-001-065)

Input Specification:
`#define NUMBER_OF_PRODUCERS 4`
`#define NUMBER_OF_CONSUMERS 65536`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The workload compiled but did not linked, as the linker could not create a binary image that could be loaded into the available amount of RAM memory.	
Notes:	

The linker error was:

```
/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-evt.exe section .bss)
collect2: ld returned 1 exit status
make: *** [o-optimize/rtems-cmp-cl-evt] Error 1
```

A.6.2. RTEMS-TS-CL-EVT-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-EVT-002
Purpose:	To stress the event manager by increasing the number of events that producer tasks generate.
Fault Location(s):	Source file: rtems-cmp-cl-evt.h Lines: [93] `#define NUMBER_OF_EVENTS 320`
Generated Test Cases:	4

A.6.2.1. Test Case Results

No faults were detected in this RTEMS task manager with the test cases defined within this test suite.

A.7. RTEMS Partition Manager Campaign

Test Campaign Definition	
Campaign Identifier:	RTEMS-CMP-CL-PRT
Purpose:	To stress test the partition manager.
Workload Files:	rtems-cmp-cl-prt.c, rtems-cmp-cl-prt.h, prt-producer.c, prt-consumer.c
Test Suites:	1. RTEMS-TS-CL-PRT-001 2. RTEMS-TS-CL-PRT-002
Workload Description:	This workload will create a partition and a message queue for each system. Then, each producer will obtain a buffer from its system partition, write its task identifier in the first bytes of this buffer and send to the message queue a structure containing a pointer to the allocated buffer. The consumers, which will be waiting in the message queue, will then receive this structure, check the producer task identifier and return the buffer again to the partition. The stressing will consist on the modification of the number of systems created and the number of threads (consumers and producers) that allocate/free partition buffers within each system. The size of the buffers allocated and the total partition size shall also be modified.

The following table presents the partition manager workload specific parameters and its default values.

Data Type	Parameter Name	Default Value	Description
rtems_unsigned32	PARTITION_SIZE	1024	The size of each partition
rtems_unsigned32	PARTITION_BUFFER_SIZE	16	The size of each partition buffer
rtems_unsigned32	NUMBER_OF_BUFFERS	100	Number of buffers to obtain by each producer

A.7.1. RTEMS-TS-CL-PRT-001

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-PRT-001
Purpose:	To stress the partition manager by creating as many systems as possible. The producer's priority shall be lower than the consumers.

Fault Location(s):	Source file: rtems-cmp-cl-prt.h
	Lines: [50, 65, 80]
	`#define NUMBER_OF_PRODUCERS 64`
	`#define NUMBER_OF_CONSUMERS 64`
	`#define NUMBER_OF_SYSTEMS 4`

Generated Test Cases:	64

A.7.1.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-001-004 (same results obtained in RTEMS-TCR-CL-PRT-001-008, RTEMS-TCR-CL-PRT-001-010, RTEMS-TCR-CL-PRT-001-011, RTEMS-TCR-CL-PRT-001-012, RTEMS-TCR-CL-PRT-001-020, RTEMS-TCR-CL-PRT-001-024, RTEMS-TCR-CL-PRT-001-026, RTEMS-TCR-CL-PRT-001-027, RTEMS-TCR-CL-PRT-001-034, RTEMS-TCR-CL-PRT-001-035, RTEMS-TCR-CL-PRT-001-038)

Input Specification:
`#define NUMBER_OF_PRODUCERS 256`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: Not enough RAM to execute the application. The simulator returned the following output: `bspstart: Not enough RAM!!!`	
Notes:	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-001-005 (same results obtained in RTEMS-TCR-CL-PRT-001-009, RTEMS-TCR-CL-PRT-001-013, RTEMS-TCR-CL-PRT-001-014, RTEMS-TCR-CL-PRT-001-015, RTEMS-TCR-CL-PRT-001-016, RTEMS-TCR-CL-PRT-001-017, RTEMS-TCR-CL-PRT-001-021, RTEMS-TCR-CL-PRT-001-025, RTEMS-TCR-CL-PRT-001-028, RTEMS-TCR-CL-PRT-001-029, RTEMS-TCR-CL-PRT-001-030, RTEMS-TCR-CL-PRT-001-031, RTEMS-TCR-CL-PRT-001-032, RTEMS-TCR-CL-PRT-001-033, RTEMS-TCR-CL-PRT-001-036, RTEMS-TCR-CL-PRT-001-039, RTEMS-TCR-CL-PRT-001-040, RTEMS-TCR-CL-PRT-001-042, RTEMS-TCR-CL-PRT-001-043, RTEMS-TCR-CL-PRT-001-044, RTEMS-TCR-CL-PRT-001-050, RTEMS-TCR-CL-PRT-001-051, RTEMS-TCR-CL-PRT-001-054, RTEMS-TCR-CL-PRT-001-055)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 4`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-001-037 (same results obtained in RTEMS-TCR-CL-PRT-001-041, RTEMS-TCR-CL-PRT-001-045, RTEMS-TCR-CL-PRT-001-046, RTEMS-TCR-CL-PRT-001-047, RTEMS-TCR-CL-PRT-001-048, RTEMS-TCR-CL-PRT-001-049, RTEMS-TCR-CL-PRT-001-052, RTEMS-TCR-CL-PRT-001-053, RTEMS-TCR-CL-PRT-001-056, RTEMS-TCR-CL-PRT-001-057, RTEMS-TCR-CL-PRT-001-058, RTEMS-TCR-CL-PRT-001-059, RTEMS-TCR-CL-PRT-001-060, RTEMS-TCR-CL-PRT-001-061, RTEMS-TCR-CL-PRT-001-062, RTEMS-TCR-CL-PRT-001-063, RTEMS-TCR-CL-PRT-001-064, RTEMS-TCR-CL-PRT-001-065)

Input Specification:
`#define NUMBER_OF_PRODUCERS 65536`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 256`	

Failure Description: The workload compiled but did not linked, as the linker could not create a binary image that could be loaded into the available amount of RAM memory.	
Notes:	

The linker error was:

```
/opt/rtems/sparc-rtems/bin/ld: region ram is full (o-optimize/rtems-cmp-cl-prt.exe section .bss)
collect2: ld returned 1 exit status
make: *** [o-optimize/rtems-cmp-cl-prt] Error 1
```

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-001-018 (same results obtained in RTEMS-TCR-CL-PRT-001-019, RTEMS-TCR-CL-PRT-001-022, RTEMS-TCR-CL-PRT-001-023)

Input Specification:
`#define NUMBER_OF_PRODUCERS 4`
`#define NUMBER_OF_CONSUMERS 4`
`#define NUMBER_OF_SYSTEMS 16`	

Failure Description: An error code was returned while creating the systems tasks. The error code, `RTEMS_UNSATISFIED`, means that there was no space left in memory for the task stack/FP context.	
Notes:	

A.7.2. RTEMS-TS-CL-PRT-002

Test Suite Definition	
Test Suite Identifier:	RTEMS-TS-CL-PRT-002
Purpose:	To stress the partition manager by using different partitions layouts (e.g., large partitions with small buffers, small partitions with large buffers, etc. Other parameter to modify is the number of buffers each producer has to take from the partition.

Fault Location(s):	Source file: rtems-cmp-cl-prt.h
	Lines: [90, 93, 96]
	`#define PARTITION_SIZE 1024`
	`#define PARTITION_BUFFER_SIZE 16`
	`#define NUMBER_OF_BUFFERS 100`

Generated Test Cases:	64

A.7.2.1. Test Case Results

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-002-120 (same results obtained in RTEMS-TCR-CL-PRT-001-121, RTEMS-TCR-CL-PRT-001-125)

Input Specification:
`#define PARTITION_SIZE 16`
`#define PARTITION_BUFFER_SIZE 16`
`#define NUMBER_OF_BUFFERS 65536`	

Failure Description: The application did not return any output. Although the application has compiled with no problems, the execution finished before the workload code have been executed.	
Notes: For further analysis, see RTEMS-TCR-CL-TSK-001-005.	

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-002-074 (same results obtained in RTEMS-TCR-CL-PRT-002-078, RTEMS-TCR-CL-PRT-002-090, RTEMS-TCR-CL-PRT-002-094, RTEMS-TCR-CL-PRT-002-106, RTEMS-TCR-CL-PRT-002-110, RTEMS-TCR-CL-PRT-002-122, RTEMS-TCR-CL-PRT-002-126)

Input Specification:

```
#define PARTITION_SIZE 65536
#define PARTITION_BUFFER_SIZE 16
#define NUMBER_OF_BUFFERS 4
```

Failure Description:

An error code was returned while creating the systems tasks. The error code, `RTEMS_UNSATISFIED`, means that there was no space left in memory for the task stack/FP context.

Notes:

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-002-067 (same results obtained in RTEMS-TCR-CL-PRT-002-068, RTEMS-TCR-CL-PRT-002-069, RTEMS-TCR-CL-PRT-002-070, RTEMS-TCR-CL-PRT-002-083, RTEMS-TCR-CL-PRT-002-084, RTEMS-TCR-CL-PRT-002-085, RTEMS-TCR-CL-PRT-002-086, RTEMS-TCR-CL-PRT-002-099, RTEMS-TCR-CL-PRT-002-100, RTEMS-TCR-CL-PRT-002-101, RTEMS-TCR-CL-PRT-002-102, RTEMS-TCR-CL-PRT-002-115, RTEMS-TCR-CL-PRT-002-116, RTEMS-TCR-CL-PRT-002-117, RTEMS-TCR-CL-PRT-002-118)

Input Specification:

```
#define PARTITION_SIZE 4
#define PARTITION_BUFFER_SIZE 4
#define NUMBER_OF_BUFFERS 4
```

Failure Description:

The `RTEMS_INVALID_SIZE` error code was returned while creating the systems partitions.

Notes:

According to the RTEMS documentation, the `RTEMS_INVALID_SIZE` error is returned in the following situations:

- when the partition length or the buffers size is less then zero;
- when the partition length is less then the buffers size, or
- when the buffer size is not a multiple of four.

However, in these test cases, the buffer size was always four and the API call is returning this error code.

Test Case Results	
Test case result identifier:	RTEMS-TCR-CL-PRT-002-071 (same results obtained in RTEMS-TCR-CL-PRT-002-075, RTEMS-TCR-CL-PRT-002-076, RTEMS-TCR-CL-PRT-002-079, RTEMS-TCR-CL-PRT-002-080, RTEMS-TCR-CL-PRT-002-081, RTEMS-TCR-CL-PRT-002-087, RTEMS-TCR-CL-PRT-002-091, RTEMS-TCR-CL-PRT-002-092, RTEMS-TCR-CL-PRT-002-095, RTEMS-TCR-CL-PRT-002-096, RTEMS-TCR-CL-PRT-002-097, RTEMS-TCR-CL-PRT-002-103, RTEMS-TCR-CL-PRT-002-107, RTEMS-TCR-CL-PRT-002-108, RTEMS-TCR-CL-PRT-002-111, RTEMS-TCR-CL-PRT-002-112, RTEMS-TCR-CL-PRT-002-113, RTEMS-TCR-CL-PRT-002-119, RTEMS-TCR-CL-PRT-002-123, RTEMS-TCR-CL-PRT-002-124, RTEMS-TCR-CL-PRT-002-127, RTEMS-TCR-CL-PRT-002-128, RTEMS-TCR-CL-PRT-002-129)

Input Specification:

```
#define PARTITION_SIZE 4
#define PARTITION_BUFFER_SIZE 16
#define NUMBER_OF_BUFFERS 4
```

Failure Description:

An error code was returned while creating the systems tasks. The error code, `RTEMS_UNSATISFIED`, is due to the fact that the buffer length is less then the buffers size.

Notes:

Annex
B

Annex B. Workloads for the RTEMS Classical API Test Suites

The following sections present the workloads that were used to build the test cases for the stress testing.

B.1. Task Manager Workload

rtems-cmp-cl-tsk.h

```
 1  /*****************************************************************************
 2   SRC-MODULE : Task Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-tsk/rtems-cmp-cl-tsk.h,v $
 6   $Id: rtems-cmp-cl-tsk.h,v 1.2 2003/10/24 13:23:34 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Task Manager.
20
21   CREATED ON : 01-09-2003
22   CHANGED ON : $Date: 2003/10/24 13:23:34 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-tsk.h,v $
35   Revision 1.2  2003/10/24 13:23:34  lhenriques
36   Changed default parameters.
37
38   Revision 1.1  2003/10/20 14:33:15  lhenriques
39   First version of the stress-testing workload for the task manager.
```

```
40
41
42   ****************************************************************************/
43
44  /*
45     * The following definitions are the parameters that shall be changed to
generated
46   * diferent loads on the target.
47   */
48
49  /* Number of producer tasks */
50  #define NUMBER_OF_PRODUCERS 64
51
52  /* Stack size for the producers tasks */
53  #define PRODUCERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)
54
55  /* Producers priority */
56  #define PRODUCERS_PRIORITY 3
57
58  /* Producers tasks mode */
59  #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
60
61  /* Producers tasks attributes */
62  #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
63
64  /* Number of consumers tasks */
65  #define NUMBER_OF_CONSUMERS 64
66
67  /* Stack size for the consumers tasks */
68  #define CONSUMERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)
69
70  /* Consumers tasks mode */
71  #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
72
73  /* Consumers tasks attributes */
74  #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
75
76  /* Consumers priority */
77  #define CONSUMERS_PRIORITY 2
78
79  /* Number of systems (producers/consumers systems) */
80  #define NUMBER_OF_SYSTEMS 4
81
82  /* Number of ticks to wait until workload finishes */
83  #define TEST_TIMEOUT 20000
84
85  /****************************************************************************/
86  /* The following definitions are workload dependent.                        */
87  /****************************************************************************/
88
89  /* Semaphores initial count */
90  #define SEMAPHORES_INITIAL_COUNT 0
91
92  /* Semaphores attributes */
93  #define SEMAPHORES_ATTRIBUTES RTEMS_DEFAULT_ATTRIBUTES
94
95  /* Semaphores priority */
96  #define SEMAPHORES_PRIORITY RTEMS_NO_PRIORITY
```

```
 97
 98  /* Number of semaphore signals for each producer */
 99  #define NUMBER_OF_PRODUCED_ITEMS 100
100
101  /* Time to wait before producing another item */
102  #define DELAY_BETWEEN_ITEMS 1
103
104  /*
105   * Enumeration with all the possible errors.
106   */
107  enum {
108    NO_ERROR,
109    RTEMS_SEMAPHORE_CREATE_ERROR,
110    RTEMS_SEMAPHORE_RELEASE_ERROR,
111    RTEMS_SEMAPHORE_OBTAIN_ERROR,
112    RTEMS_TASK_CREATE_ERROR,
113    RTEMS_TASK_START_ERROR,
114    RTEMS_TASK_WAKE_AFTER_ERROR,
115    RTEMS_TASK_IDENT_ERROR
116  };
117
118  /* Arrays to store the producers and consumers IDs */
119  extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
120  extern rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
121
122  /* Array containing the semaphores for all systems */
123  extern rtems_id semaphores_array[NUMBER_OF_SYSTEMS];
124
125  /* Semaphore that shall be obtained in order to exit the workload */
126  extern rtems_id results_sem;
127
128  /*
129   * These are the producer and consumer tasks.
130   */
131  rtems_task producer_task (rtems_task_argument producer_arg);
132  rtems_task consumer_task (rtems_task_argument consumer_arg);
133
134  /*
135   * This function shall analyse the results in each task,
136   */
137  void analyse_tasks_results ();
138
139  /*
140   * This function simply checks the errors in the test
141   */
142       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
143
144  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
145
146  /*
147   * This function switchs the return status parameter and prints a string with
148   * the corresponding error code.
149   */
150  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-tsk.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Task Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-tsk/rtems-cmp-cl-tsk.c,v $
 6   $Id: rtems-cmp-cl-tsk.c,v 1.2 2003/10/24 13:23:27 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Task Manager.
20
21   CREATED ON : 30-09-2003
22   CHANGED ON : $Date: 2003/10/24 13:23:27 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-tsk.c,v $
35   Revision 1.2  2003/10/24 13:23:27  lhenriques
36   Corrected some problems in the tasks analysis.
37
38   Revision 1.1  2003/10/20 14:33:15  lhenriques
39   First version of the stress-testing workload for the task manager.
40
41
42   *****************************************************************************/
43
44  #include <bsp.h>
45  #include <stdio.h>
46  #include "rtems-cmp-cl-tsk.h"
47
48  rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
49  rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
50
51  rtems_id semaphores_array[NUMBER_OF_SYSTEMS];
52
53  rtems_id results_sem;
54
55  int created_producers = 0;
```

```
 56 int created_consumers = 0;
 57
 58 /*
 59  * Fatal error handler. It is used to find out whether an halt has
 60  * occurred during the workload execution.
 61  */
 62 rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
 63                                      boolean is_internal,
 64                                      rtems_unsigned32 the_error)
 65 {
 66   printf ("A Fatal error has occurred!\n");
 67   printf ("Source: ");
 68   /* Find source */
 69   switch (the_source)
 70     {
 71     case INTERNAL_ERROR_CORE:
 72       printf ("INTERNAL_ERROR_CORE");
 73       break;
 74     case INTERNAL_ERROR_RTEMS_API:
 75       printf ("INTERNAL_ERROR_RTEMS_API");
 76       break;
 77     case INTERNAL_ERROR_POSIX_API:
 78       printf ("INTERNAL_ERROR_POSIX_API");
 79       break;
 80     case INTERNAL_ERROR_ITRON_API:
 81       printf ("INTERNAL_ERROR_ITRON_API");
 82     default:
 83       printf ("UNKOWN (%d)", the_source);
 84     }
 85   printf ("\n");
 86
 87   if (is_internal == TRUE)
 88     {
 89       printf ("It is an INTERNAL error.\n");
 90
 91       printf ("Error: ");
 92       /* Find the error itself */
 93       switch (the_error)
 94         {
 95         case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
 96           printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
 97           break;
 98         case INTERNAL_ERROR_NO_CPU_TABLE:
 99           printf ("INTERNAL_ERROR_NO_CPU_TABLE");
100           break;
101         case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
102           printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
103           break;
104         case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
105           printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
106           break;
107         case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
108           printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
109           break;
110         case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
111           printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
112           break;
113         case INTERNAL_ERROR_THREAD_EXITTED:
```

```
114             printf ("INTERNAL_ERROR_THREAD_EXITTED");
115             break;
116           case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
117             printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
118             break;
119           case INTERNAL_ERROR_INVALID_NODE:
120             printf ("INTERNAL_ERROR_INVALID_NODE");
121             break;
122           case INTERNAL_ERROR_NO_MPCI:
123             printf ("INTERNAL_ERROR_NO_MPCI");
124             break;
125           case INTERNAL_ERROR_BAD_PACKET:
126             printf ("INTERNAL_ERROR_BAD_PACKET");
127             break;
128           case INTERNAL_ERROR_OUT_OF_PACKETS:
129             printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
130             break;
131           case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
132             printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
133             break;
134           case INTERNAL_ERROR_OUT_OF_PROXIES:
135             printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
136             break;
137           case INTERNAL_ERROR_INVALID_GLOBAL_ID:
138             printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
139             break;
140           case INTERNAL_ERROR_BAD_STACK_HOOK:
141             printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
142             break;
143           case INTERNAL_ERROR_BAD_ATTRIBUTES:
144             printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
145             break;
146           default:
147             printf ("UNKNOWN (%d)", the_error);
148           }
149         printf ("\n");
150
151       }
152     else
153       {
154         printf ("It is NOT an internal error.\n");
155         /* Assume an RTEMS Classic API error... */
156         show_error_code (the_error);
157       }
158
159     analyse_tasks_results ();
160     exit (-1);
161   }
162
163   /*
164    * This structure defines the user extensions entry points
165    */
166   rtems_extensions_table user_extensions =
167     {
168       NULL,               /* task creation extension */
169       NULL,               /* task start extension */
170       NULL,               /* task restart extension */
171       NULL,               /* task delete extension */
```

```
172      NULL,               /* task switch extension */
173      NULL,               /* task begin extension */
174      NULL,               /* task exitted extension */
175      fatal_error_handler /* fatal error extension */
176    };
177
178
179  /*
180   * Init task is the first to be executed.
181   */
182  rtems_task Init(rtems_task_argument ignored)
183  {
184    rtems_unsigned32 error = NO_ERROR;
185    rtems_status_code return_status = RTEMS_NOT_DEFINED;
186    rtems_unsigned32 system_index;
187    rtems_unsigned32 index;
188    rtems_unsigned32 max_index;
189    rtems_name results_sem_name;
190    rtems_name table_name;
191    rtems_id table_id;
192    /* Add user extension table to handle fata error */
193    table_name = rtems_build_name ('U', 'S', 'E', 'R');
194    return_status = rtems_extension_create (table_name,
195                                            &user_extensions,
196                                            &table_id);
197
198    if (return_status != RTEMS_SUCCESSFUL)
199      {
200              printf ("Error registering an user handler to the fatal error
extension.\n");
201        show_error_code (return_status);
202        exit (-1);
203      }
204
205    /* Create the results semaphore */
206    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
207    return_status = rtems_semaphore_create (results_sem_name,
208                                            1, /* Only one task can exit */
209                                            RTEMS_DEFAULT_ATTRIBUTES,
210                                            RTEMS_NO_PRIORITY,
211                                            &results_sem);
212    /* If this fails... exit */
213    if (return_status != RTEMS_SUCCESSFUL)
214      {
215        printf ("Error creating semaphore for exit point.\n");
216        show_error_code (return_status);
217        exit (-1);
218      }
219
220    /* Create all the NUMBER_OF_SYSTEMS systems. */
221    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
222      {
223        rtems_name producer_task_name;
224        rtems_name consumer_task_name;
225        rtems_name sem_name;
226        /* Create the resource -- a semaphore for each system */
227        sem_name = system_index + 1;
228        return_status = rtems_semaphore_create (sem_name,
```

```
229                                             SEMAPHORES_INITIAL_COUNT,
230                                             SEMAPHORES_ATTRIBUTES,
231                                             SEMAPHORES_PRIORITY,
232                                             &semaphores_array[system_index]);
233
234         if (return_status != RTEMS_SUCCESSFUL)
235           {
236             rtems_semaphore_obtain (results_sem,
237                                     RTEMS_WAIT,
238                                     RTEMS_NO_TIMEOUT);
239             error = RTEMS_SEMAPHORE_CREATE_ERROR;
240             show_test_results (error, return_status);
241             exit (-1);
242           }
243
244         /* Create all the producers and consumers for this system */
245                   max_index  =  (NUMBER_OF_PRODUCERS  <=  NUMBER_OF_CONSUMERS  ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
246         for (index = 0; index < max_index; index++)
247           {
248             if (index < NUMBER_OF_PRODUCERS)
249               {
250                 /* Create a producer */
251                 producer_task_name = index + 1;
252                 return_status = rtems_task_create (producer_task_name,
253                                                    PRODUCERS_PRIORITY,
254                                                    PRODUCERS_TASK_STACK_SIZE,
255                                                    PRODUCERS_TASK_MODE,
256                                                    PRODUCERS_TASK_ATTR,
257
&producers_array[index][system_index]);
258                 if (return_status != RTEMS_SUCCESSFUL)
259                   {
260                     rtems_semaphore_obtain (results_sem,
261                                             RTEMS_WAIT,
262                                             RTEMS_NO_TIMEOUT);
263                     error = RTEMS_TASK_CREATE_ERROR;
264                     show_test_results (error, return_status);
265                     exit (-1);
266                   }
267
268                                                 return_status   =   rtems_task_start
(producers_array[index][system_index],
269                                                    producer_task,
270                                                    system_index);
271                 if (return_status != RTEMS_SUCCESSFUL)
272                   {
273                     rtems_semaphore_obtain (results_sem,
274                                             RTEMS_WAIT,
275                                             RTEMS_NO_TIMEOUT);
276                     error = RTEMS_TASK_START_ERROR;
277                     show_test_results (error, return_status);
278                     exit (-1);
279                   }
280                 created_producers++;
281               }
282             if (index < NUMBER_OF_CONSUMERS)
283               {
```

```
   284               /* Create a consumer */
   285               consumer_task_name = index + 1;
   286               return_status = rtems_task_create (consumer_task_name,
   287                                                  CONSUMERS_PRIORITY,
   288                                                  CONSUMERS_TASK_STACK_SIZE,
   289                                                  CONSUMERS_TASK_MODE,
   290                                                  CONSUMERS_TASK_ATTR,
   291
&consumers_array[index][system_index]);
   292               if (return_status != RTEMS_SUCCESSFUL)
   293                 {
   294                   rtems_semaphore_obtain (results_sem,
   295                                           RTEMS_WAIT,
   296                                           RTEMS_NO_TIMEOUT);
   297                   error = RTEMS_TASK_CREATE_ERROR;
   298                   show_test_results (error, return_status);
   299                   exit (-1);
   300                 }
   301
   302                                                 return_status   =   rtems_task_start
(consumers_array[index][system_index],
   303                                                    consumer_task,
   304                                                    system_index);
   305               if (return_status != RTEMS_SUCCESSFUL)
   306                 {
   307                   rtems_semaphore_obtain (results_sem,
   308                                           RTEMS_WAIT,
   309                                           RTEMS_NO_TIMEOUT);
   310                   error = RTEMS_TASK_START_ERROR;
   311                   show_test_results (error, return_status);
   312                   exit (-1);
   313                 }
   314               created_consumers++;
   315             }
   316         }
   317     }
   318
   319   /* Wait for all tasks or for an error. */
   320   return_status = rtems_task_wake_after (TEST_TIMEOUT);
   321
   322   if (return_status != RTEMS_SUCCESSFUL)
   323     {
   324       error = RTEMS_TASK_WAKE_AFTER_ERROR;
   325     }
   326
   327   rtems_semaphore_obtain (results_sem,
   328                           RTEMS_WAIT,
   329                           RTEMS_NO_TIMEOUT);
   330   show_test_results (error, return_status);
   331
   332   exit (0);
   333 }
   334
   335 void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
   336 {
   337   printf ("=============================================\n");
   338   printf ("Test Paramenters\n");
   339   printf ("=============================================\n");
```

```
340    printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
341    printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
342    printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
343    printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
344    printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
345    printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
346    printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
347    printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
348    printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
349    printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
350    printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
351    printf ("================================================\n");
352
353    if (error == NO_ERROR)
354      {
355        printf ("Systems created sucessfully:\n");
356      }
357    else
358      {
359        parse_error (error, return_status);
360      }
361    analyse_tasks_results ();
362  }
363
364  void analyse_tasks_results ()
365  {
366    int system_index, index;
367    rtems_unsigned32 running_consumers = 0;
368    rtems_unsigned32 running_producers = 0;
369    rtems_unsigned32 failed_consumers = 0;
370    rtems_unsigned32 failed_producers = 0;
371    int failed_consumer_index = -1;
372    int failed_consumer_system = -1;
373    int failed_producer_index = -1;
374    int failed_producer_system = -1;
375
376    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
377      {
378        /* Analyse producers array */
379          for (index = 0; (index < NUMBER_OF_PRODUCERS) && (((system_index + 1) *
(index + 1) + running_producers) < creat
ed_producers); index++)
380           {
381             if (producers_array[index][system_index] > 0)
382               {
383                 running_producers++;
384               }
385             else if (producers_array[index][system_index] < 0)
386               {
387                 failed_producers++;
388                 if (failed_producer_index == -1)
389                   {
390                     failed_producer_index = index;
391                     failed_producer_system = system_index;
392                   }
393               }
394           }
395        /* Analyse consumers array */
```

```
396         for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
397           {
398             if (consumers_array[index][system_index] > 0)
399               {
400                 running_consumers++;
401               }
402             else if (consumers_array[index][system_index] < 0)
403               {
404                 failed_consumers++;
405                 if (failed_consumer_index == -1)
406                   {
407                     failed_consumer_index = index;
408                     failed_consumer_system = system_index;
409                   }
410               }
411           }
412       }
413           printf ("Total number of successfully created producers: %d\n",
created_producers);
414     printf ("Running producers: %d\n", running_producers);
415      printf ("Producers that terminated successfuly: %d\n", (created_producers -
running_producers));
416     printf ("Number of failed producers: %d\n", failed_producers);
417     if (failed_producers > 0)
418       {
419               printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
420                printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
421       }
422           printf ("Total number of successfully created consumers: %d\n",
created_consumers);
423     printf ("Number of failed consumers: %d\n", failed_consumers);
424     if (failed_consumers > 0)
425       {
426               printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
427                printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
428       }
429   }
430
431   void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
432   {
433         switch (error)
434           {
435           case RTEMS_SEMAPHORE_CREATE_ERROR:
436             printf ("Error creating semaphore.\n");
437             break;
438           case RTEMS_SEMAPHORE_RELEASE_ERROR:
439             printf ("Error signaling a semaphore.\n");
440             break;
441           case RTEMS_SEMAPHORE_OBTAIN_ERROR:
442             printf ("Error obtaining a semaphore.\n");
443             break;
444           case RTEMS_TASK_CREATE_ERROR:
445             printf ("Error creating task.\n");
446             break;
```

```
447           case RTEMS_TASK_START_ERROR:
448             printf ("Error starting task.\n");
449             break;
450           case RTEMS_TASK_WAKE_AFTER_ERROR:
451             printf ("Error waiting for tasks to finish.\n");
452             break;
453           case RTEMS_TASK_IDENT_ERROR:
454             printf ("Error obtaining thread ID.\n");
455             break;
456           default:
457             printf ("Unknown failure. Possible bug in workload.\n");
458             break;
459           }
460         show_error_code (return_status);
461   }
462
463   void show_error_code (rtems_status_code return_status)
464   {
465     printf ("Error code: ");
466     switch (return_status)
467       {
468       case RTEMS_TASK_EXITTED:
469         printf ("RTEMS_TASK_EXITTED");
470         break;
471       case RTEMS_MP_NOT_CONFIGURED:
472         printf ("RTEMS_MP_NOT_CONFIGURED");
473         break;
474       case RTEMS_INVALID_NAME:
475         printf ("RTEMS_INVALID_NAME");
476         break;
477       case RTEMS_INVALID_ID:
478         printf ("RTEMS_INVALID_ID");
479         break;
480       case RTEMS_TOO_MANY:
481         printf ("RTEMS_TOO_MANY");
482         break;
483       case RTEMS_TIMEOUT:
484         printf ("RTEMS_TIMEOUT");
485         break;
486       case RTEMS_OBJECT_WAS_DELETED:
487         printf ("RTEMS_OBJECT_WAS_DELETED");
488         break;
489       case RTEMS_INVALID_SIZE:
490         printf ("RTEMS_INVALID_SIZE");
491         break;
492       case RTEMS_INVALID_ADDRESS:
493         printf ("RTEMS_INVALID_ADDRESS");
494         break;
495       case RTEMS_INVALID_NUMBER:
496         printf ("RTEMS_INVALID_NUMBER");
497         break;
498       case RTEMS_NOT_DEFINED:
499         printf ("RTEMS_NOT_DEFINED");
500         break;
501       case RTEMS_RESOURCE_IN_USE:
502         printf ("RTEMS_RESOURCE_IN_USE");
503         break;
504       case RTEMS_UNSATISFIED:
```

```
505       printf ("RTEMS_UNSATISFIED");
506       break;
507     case RTEMS_INCORRECT_STATE:
508       printf ("RTEMS_INCORRECT_STATE");
509       break;
510     case RTEMS_ALREADY_SUSPENDED:
511       printf ("RTEMS_ALREADY_SUSPENDED");
512       break;
513     case RTEMS_ILLEGAL_ON_SELF:
514       printf ("RTEMS_ILLEGAL_ON_SELF");
515       break;
516     case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
517       printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
518       break;
519     case RTEMS_CALLED_FROM_ISR:
520       printf ("RTEMS_CALLED_FROM_ISR");
521       break;
522     case RTEMS_INVALID_PRIORITY:
523       printf ("RTEMS_INVALID_PRIORITY");
524       break;
525     case RTEMS_INVALID_CLOCK:
526       printf ("RTEMS_INVALID_CLOCK");
527       break;
528     case RTEMS_INVALID_NODE:
529       printf ("RTEMS_INVALID_NODE");
530       break;
531     case RTEMS_NOT_CONFIGURED:
532       printf ("RTEMS_NOT_CONFIGURED");
533       break;
534     case RTEMS_NOT_OWNER_OF_RESOURCE:
535       printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
536       break;
537     case RTEMS_NOT_IMPLEMENTED:
538       printf ("RTEMS_NOT_IMPLEMENTED");
539       break;
540     case RTEMS_INTERNAL_ERROR:
541       printf ("RTEMS_INTERNAL_ERROR");
542       break;
543     case RTEMS_NO_MEMORY:
544       printf ("RTEMS_NO_MEMORY");
545       break;
546     case RTEMS_IO_ERROR:
547       printf ("RTEMS_IO_ERROR");
548       break;
549     case RTEMS_PROXY_BLOCKING:
550       printf ("RTEMS_PROXY_BLOCKING");
551       break;
552     default:
553       printf ("UNKNOWN ERROR CODE");
554     }
555   printf ("\n");
556 }
557
558 /* configuration information */
559
560 #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
561 #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
562
```

```
563  #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
564   #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
565
566  #define CONFIGURE_MAXIMUM_SEMAPHORES (NUMBER_OF_SYSTEMS + 1)
567
568  #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
569
570  #define CONFIGURE_INIT
571
572  #include <confdefs.h>
573
574  /* end of file */
```

tsk-producer.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Task Manager Workload
 3   MODULE-VERS : N/A
 4
 5           $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-tsk/tsk-producer.c,v $
 6   $Id: tsk-producer.c,v 1.1 2003/10/20 14:33:15 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Task
20   Manager. This file contains the producer task for this workload.
21
22   CREATED ON : 30-09-2003
23   CHANGED ON : $Date: 2003/10/20 14:33:15 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: tsk-producer.c,v $
36   Revision 1.1  2003/10/20 14:33:15  lhenriques
37   First version of the stress-testing workload for the task manager.
38
39
```

```
40   ****************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-tsk.h"
45
46  void set_producer_result (int res, rtems_id task_id, rtems_unsigned32 system)
47  {
48    int i;
49    for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
50      {
51        if (producers_array[i][system] == task_id)
52          {
53            producers_array[i][system] = res;
54            return;
55          }
56      }
57    printf ("[ERROR] Could not obtain the producer %d index in producers
58    array.\n", task_id);
59  }
60
61  /*
62   * This task will execute a loop where the system resource (a semaphore) will be
63   * signalled several times (NUMBER_OF_PRODUCED_ITEMS). Between every signal, the
64   * task sleep for DELAY_BETWEEN_ITEMS time.
65   * If an error occurs, the task tries to obtains the results_sem, shows the test
66   * results and exists.
67   */
68  rtems_task producer_task (rtems_task_argument producer_arg)
69  {
70    /* Get system number index the semaphores array */
71    rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
72    /* Number of rtems_semaphore_release calls */
73    rtems_unsigned32 count = NUMBER_OF_PRODUCED_ITEMS;
74    /* Get the semaphore */
75    rtems_id sem_id = semaphores_array[system];
76    rtems_status_code return_status = RTEMS_NOT_DEFINED;
77    rtems_unsigned32 error = NO_ERROR;
78    rtems_id task_id;
79
80    return_status = rtems_task_ident (RTEMS_SELF,
81                                      RTEMS_SEARCH_ALL_NODES,
82                                      &task_id);
83    if (return_status != RTEMS_SUCCESSFUL)
84      {
85        rtems_semaphore_obtain (results_sem,
86                                RTEMS_WAIT,
87                                RTEMS_NO_TIMEOUT);
88        error = RTEMS_TASK_IDENT_ERROR;
89        show_test_results (error, return_status);
90        analyse_tasks_results ();
91        exit (-1);
92      }
93
94    while (count-- > 0)
95      {
96        return_status = rtems_semaphore_release (sem_id);
97        if (return_status != RTEMS_SUCCESSFUL)
```

```
 98          {
 99            rtems_semaphore_obtain (results_sem,
100                                    RTEMS_WAIT,
101                                    RTEMS_NO_TIMEOUT);
102            set_producer_result (-1, task_id, system);
103            error = RTEMS_SEMAPHORE_RELEASE_ERROR;
104            show_test_results (error, return_status);
105            exit (-1);
106          }
107        /* Wait before signaling again the semaphore */
108        return_status = rtems_task_wake_after (DELAY_BETWEEN_ITEMS);
109        if (return_status != RTEMS_SUCCESSFUL)
110          {
111            rtems_semaphore_obtain (results_sem,
112                                    RTEMS_WAIT,
113                                    RTEMS_NO_TIMEOUT);
114            set_producer_result (-1, task_id, system);
115            error = RTEMS_TASK_WAKE_AFTER_ERROR;
116            show_test_results (error, return_status);
117            exit (-1);
118          }
119      }
120    set_producer_result (0, task_id, system);
121  }
```

tsk-consumer.c

```
  1  /*****************************************************************************
  2   SRC-MODULE : Task Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-tsk/tsk-consumer.c,v $
  6   $Id: tsk-consumer.c,v 1.1 2003/10/20 14:33:15 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Task
 20   Manager. This file contains the consumer task for this workload.
 21
 22   CREATED ON : 30-09-2003
 23   CHANGED ON : $Date: 2003/10/20 14:33:15 $
 24   CHANGED BY : $Author: lhenriques $
 25
 26   $Revision: 1.1 $
 27   STICKY TAG : $Name:  $
 28
 29   INSPECTED ON:
```

```
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: tsk-consumer.c,v $
36   Revision 1.1  2003/10/20 14:33:15  lhenriques
37   First version of the stress-testing workload for the task manager.
38
39
40   ******************************************************************************/
41
42 #include <bsp.h>
43 #include <stdio.h>
44 #include "rtems-cmp-cl-tsk.h"
45
46 void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
47 {
48   int i;
49   for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
50     {
51       if (consumers_array[i][system] == task_id)
52         {
53           consumers_array[i][system] = -1;
54           return;
55         }
56     }
57   printf ("[ERROR] Could not obtain the consumer %d index in consumers
58   array.\n", task_id);
59 }
60
61 /*
62  * This task will wait forever in the resource (semaphore). It exits only if an
63  * error occurs in the return status of the wait directive.
64  */
65 rtems_task consumer_task (rtems_task_argument consumer_arg)
66 {
67   /* Get system number index the semaphores array */
68   rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
69   /* Get the semaphore */
70   rtems_id sem_id = semaphores_array[system];
71   rtems_status_code return_status = RTEMS_NOT_DEFINED;
72   rtems_unsigned32 error = NO_ERROR;
73   rtems_id task_id;
74
75   return_status = rtems_task_ident (RTEMS_SELF,
76                                     RTEMS_SEARCH_ALL_NODES,
77                                     &task_id);
78   if (return_status != RTEMS_SUCCESSFUL)
79     {
80       rtems_semaphore_obtain (results_sem,
81                               RTEMS_WAIT,
82                               RTEMS_NO_TIMEOUT);
83       error = RTEMS_TASK_IDENT_ERROR;
84       show_test_results (error, return_status);
85       exit (-1);
86     }
87   while (TRUE)
```

```
 88      {
 89        return_status = rtems_semaphore_obtain (sem_id,
 90                                                RTEMS_WAIT,
 91                                                RTEMS_NO_TIMEOUT);
 92        if (return_status != RTEMS_SUCCESSFUL)
 93          {
 94            rtems_semaphore_obtain (results_sem,
 95                                    RTEMS_WAIT,
 96                                    RTEMS_NO_TIMEOUT);
 97            set_consumer_error (task_id, system);
 98            error = RTEMS_SEMAPHORE_OBTAIN_ERROR;
 99            show_test_results (error, return_status);
100            exit (-1);
101          }
102      }
103  }
```

B.2. Semaphore Manager Workload

rtems-cmp-cl-smp.h

```
  1  /*************************************************************************
  2   SRC-MODULE : Semaphore Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-smp/rtems-cmp-cl-smp.h,v $
  6   $Id: rtems-cmp-cl-smp.h,v 1.3 2003/10/24 13:22:02 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Semaphore Manager.
 20
 21   CREATED ON : 01-09-2003
 22   CHANGED ON : $Date: 2003/10/24 13:22:02 $
 23   CHANGED BY : $Author: lhenriques $
 24
 25   $Revision: 1.3 $
 26   STICKY TAG : $Name:  $
 27
 28   INSPECTED ON:
 29   MODERATOR :
 30
 31   TABLES : none.
 32
 33   HISTORY
 34   $Log: rtems-cmp-cl-smp.h,v $
```

```
 35   Revision 1.3  2003/10/24 13:22:02  lhenriques
 36   Changed default parameters.
 37
 38   Revision 1.2  2003/10/01 17:08:16  lhenriques
 39   Added a Fatal Error user extension handler and the headings to the header file.
 40
 41
 42   ******************************************************************************/
 43
 44  /*
 45     * The following definitions are the parameters that shall be changed to
generated
 46   * diferent loads on the target.
 47   */
 48
 49  /* Number of producer tasks */
 50  #define NUMBER_OF_PRODUCERS 64
 51
 52  /* Stack size for the producers tasks */
 53  #define PRODUCERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)
 54
 55  /* Producers priority */
 56  #define PRODUCERS_PRIORITY 2
 57
 58  /* Producers tasks mode */
 59  #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
 60
 61  /* Producers tasks attributes */
 62  #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 63
 64  /* Number of consumers tasks */
 65  #define NUMBER_OF_CONSUMERS 64
 66
 67  /* Stack size for the consumers tasks */
 68  #define CONSUMERS_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 1)
 69
 70  /* Consumers tasks mode */
 71  #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
 72
 73  /* Consumers tasks attributes */
 74  #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 75
 76  /* Consumers priority */
 77  #define CONSUMERS_PRIORITY 3
 78
 79  /* Number of systems (producers/consumers systems) */
 80  #define NUMBER_OF_SYSTEMS 4
 81
 82  /* Number of ticks to wait until workload finishes */
 83  #define TEST_TIMEOUT 20000
 84
 85  /******************************************************************************/
 86  /* The following definitions are workload dependent.                          */
 87  /******************************************************************************/
 88
 89  /* Semaphores initial count */
 90  #define SEMAPHORES_INITIAL_COUNT 0
 91
```

```
 92  /* Semaphores attributes */
 93  #define SEMAPHORES_ATTRIBUTES RTEMS_DEFAULT_ATTRIBUTES
 94
 95  /* Semaphores priority */
 96  #define SEMAPHORES_PRIORITY RTEMS_NO_PRIORITY
 97
 98  /* Number of semaphore signals for each producer */
 99  #define NUMBER_OF_PRODUCED_ITEMS 1000
100
101  /* Time to wait before producing another item */
102  #define DELAY_BETWEEN_ITEMS 1
103
104  /*
105   * Enumeration with all the possible errors.
106   */
107  enum {
108    NO_ERROR,
109    RTEMS_SEMAPHORE_CREATE_ERROR,
110    RTEMS_SEMAPHORE_RELEASE_ERROR,
111    RTEMS_SEMAPHORE_OBTAIN_ERROR,
112    RTEMS_TASK_CREATE_ERROR,
113    RTEMS_TASK_START_ERROR,
114    RTEMS_TASK_WAKE_AFTER_ERROR,
115    RTEMS_TASK_IDENT_ERROR
116  };
117
118  /* Arrays to store the producers and consumers IDs */
119  extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
120  extern rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
121
122  /* Array containing the semaphores for all systems */
123  extern rtems_id semaphores_array[NUMBER_OF_SYSTEMS];
124
125  /* Semaphore that shall be obtained in order to exit the workload */
126  extern rtems_id results_sem;
127
128  /*
129   * These are the producer and consumer tasks.
130   */
131  rtems_task producer_task (rtems_task_argument producer_arg);
132  rtems_task consumer_task (rtems_task_argument consumer_arg);
133
134  /*
135   * This function shall analyse the results in each task,
136   */
137  void analyse_tasks_results ();
138
139  /*
140   * This function simply checks the errors in the test
141  H */
142       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
143
144  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
145
146  /*
147   * This function switchs the return status parameter and prints a string with
148   * the corresponding error code.
```

```
149   */
150  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-smp.c

```
  1  /*************************************************************************
  2   SRC-MODULE : Semaphore Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-smp/rtems-cmp-cl-smp.c,v $
  6   $Id: rtems-cmp-cl-smp.c,v 1.3 2003/10/24 13:21:54 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Semaphore Manager.
 20
 21   CREATED ON : 30-09-2003
 22   CHANGED ON : $Date: 2003/10/24 13:21:54 $
 23   CHANGED BY : $Author: lhenriques $
 24
 25   $Revision: 1.3 $
 26   STICKY TAG : $Name:  $
 27
 28   INSPECTED ON:
 29   MODERATOR :
 30
 31   TABLES : none.
 32
 33   HISTORY
 34   $Log: rtems-cmp-cl-smp.c,v $
 35   Revision 1.3  2003/10/24 13:21:54  lhenriques
 36   Corrected some problems in the tasks analysis.
 37
 38   Revision 1.2  2003/10/01 17:08:16  lhenriques
 39   Added a Fatal Error user extension handler and the headings to the header file.
 40
 41   Revision 1.1  2003/10/01 14:56:39  lhenriques
 42   First version of the stress-testing workload for the semaphore manager.
 43
 44
 45   *************************************************************************/
 46
 47  #include <bsp.h>
 48  #include <stdio.h>
 49  #include "rtems-cmp-cl-smp.h"
 50
 51  rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
```

```
 52  rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
 53
 54  rtems_id semaphores_array[NUMBER_OF_SYSTEMS];
 55
 56  rtems_id results_sem;
 57
 58  int created_producers = 0;
 59  int created_consumers = 0;
 60
 61  /*
 62   * Fatal error handler. It is used to find out whether an halt has
 63   * occurred during the workload execution.
 64   */
 65  rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
 66                                       boolean is_internal,
 67                                       rtems_unsigned32 the_error)
 68  {
 69    printf ("A Fatal error has occurred!\n");
 70    printf ("Source: ");
 71    /* Find source */
 72    switch (the_source)
 73      {
 74      case INTERNAL_ERROR_CORE:
 75        printf ("INTERNAL_ERROR_CORE");
 76        break;
 77      case INTERNAL_ERROR_RTEMS_API:
 78        printf ("INTERNAL_ERROR_RTEMS_API");
 79        break;
 80      case INTERNAL_ERROR_POSIX_API:
 81        printf ("INTERNAL_ERROR_POSIX_API");
 82        break;
 83      case INTERNAL_ERROR_ITRON_API:
 84        printf ("INTERNAL_ERROR_ITRON_API");
 85      default:
 86        printf ("UNKOWN (%d)", the_source);
 87      }
 88    printf ("\n");
 89
 90    if (is_internal == TRUE)
 91      {
 92        printf ("It is an INTERNAL error.\n");
 93
 94        printf ("Error: ");
 95        /* Find the error itself */
 96        switch (the_error)
 97          {
 98          case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
 99            printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
100            break;
101          case INTERNAL_ERROR_NO_CPU_TABLE:
102            printf ("INTERNAL_ERROR_NO_CPU_TABLE");
103            break;
104          case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
105            printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
106            break;
107          case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
108            printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
109            break;
```

```
110           case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
111             printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
112             break;
113           case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
114             printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
115             break;
116           case INTERNAL_ERROR_THREAD_EXITTED:
117             printf ("INTERNAL_ERROR_THREAD_EXITTED");
118             break;
119           case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
120             printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
121             break;
122           case INTERNAL_ERROR_INVALID_NODE:
123             printf ("INTERNAL_ERROR_INVALID_NODE");
124             break;
125           case INTERNAL_ERROR_NO_MPCI:
126             printf ("INTERNAL_ERROR_NO_MPCI");
127             break;
128           case INTERNAL_ERROR_BAD_PACKET:
129             printf ("INTERNAL_ERROR_BAD_PACKET");
130             break;
131           case INTERNAL_ERROR_OUT_OF_PACKETS:
132             printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
133             break;
134           case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
135             printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
136             break;
137           case INTERNAL_ERROR_OUT_OF_PROXIES:
138             printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
139             break;
140           case INTERNAL_ERROR_INVALID_GLOBAL_ID:
141             printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
142             break;
143           case INTERNAL_ERROR_BAD_STACK_HOOK:
144             printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
145             break;
146           case INTERNAL_ERROR_BAD_ATTRIBUTES:
147             printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
148             break;
149           default:
150             printf ("UNKNOWN (%d)", the_error);
151           }
152         printf ("\n");
153
154       }
155     else
156       {
157         printf ("It is NOT an internal error.\n");
158         /* Assume an RTEMS Classic API error... */
159         show_error_code (the_error);
160       }
161     analyse_tasks_results ();
162     exit (-1);
163   }
164
165   /*
166    * This structure defines the user extensions entry points
167    */
```

```
168  rtems_extensions_table user_extensions =
169    {
170      NULL,               /* task creation extension */
171      NULL,               /* task start extension */
172      NULL,               /* task restart extension */
173      NULL,               /* task delete extension */
174      NULL,               /* task switch extension */
175      NULL,               /* task begin extension */
176      NULL,               /* task exitted extension */
177      fatal_error_handler /* fatal error extension */
178    };
179
180
181  /*
182   * Init task is the first to be executed.
183   */
184  rtems_task Init(rtems_task_argument ignored)
185  {
186    rtems_unsigned32 error = NO_ERROR;
187    rtems_status_code return_status = RTEMS_NOT_DEFINED;
188    rtems_unsigned32 system_index;
189    rtems_unsigned32 index;
190    rtems_unsigned32 max_index;
191    rtems_name results_sem_name;
192    rtems_name table_name;
193    rtems_id table_id;
194
195    /* Add user extension table to handle fata error */
196    table_name = rtems_build_name ('U', 'S', 'E', 'R');
197    return_status = rtems_extension_create (table_name,
198                                            &user_extensions,
199                                            &table_id);
200
201    if (return_status != RTEMS_SUCCESSFUL)
202      {
203              printf  ("Error  registering  an  user  handler  to  the  fatal  error
extension.\n");
204        show_error_code (return_status);
205        exit (-1);
206      }
207
208    /* Create the results semaphore */
209    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
210    return_status = rtems_semaphore_create (results_sem_name,
211                                            1, /* Only one task can exit */
212                                            RTEMS_DEFAULT_ATTRIBUTES,
213                                            RTEMS_NO_PRIORITY,
214                                            &results_sem);
215    /* If this fails... exit */
216    if (return_status != RTEMS_SUCCESSFUL)
217      {
218        printf ("Error creating semaphore for exit point.\n");
219        show_error_code (return_status);
220        exit (-1);
221      }
222
223    /* Create all the NUMBER_OF_SYSTEMS systems. */
224    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
```

```
225     {
226       rtems_name producer_task_name;
227       rtems_name consumer_task_name;
228       rtems_name sem_name;
229
230       /* Create the resource -- a semaphore for each system */
231       sem_name = system_index + 1;
232       return_status = rtems_semaphore_create (sem_name,
233                                               SEMAPHORES_INITIAL_COUNT,
234                                               SEMAPHORES_ATTRIBUTES,
235                                               SEMAPHORES_PRIORITY,
236                                               &semaphores_array[system_index]);
237
238       if (return_status != RTEMS_SUCCESSFUL)
239         {
240           rtems_semaphore_obtain (results_sem,
241                                   RTEMS_WAIT,
242                                   RTEMS_NO_TIMEOUT);
243           error = RTEMS_SEMAPHORE_CREATE_ERROR;
244           show_test_results (error, return_status);
245           exit (-1);
246         }
247
248       /* Create all the producers and consumers for this system */
249                 max_index  =  (NUMBER_OF_PRODUCERS  <=  NUMBER_OF_CONSUMERS  ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
250       for (index = 0; index < max_index; index++)
251         {
252           if (index < NUMBER_OF_PRODUCERS)
253             {
254               /* Create a producer */
255               producer_task_name = index + 1;
256               return_status = rtems_task_create (producer_task_name,
257                                                  PRODUCERS_PRIORITY,
258                                                  PRODUCERS_TASK_STACK_SIZE,
259                                                  PRODUCERS_TASK_MODE,
260                                                  PRODUCERS_TASK_ATTR,
261
&producers_array[index][system_index]);
262               if (return_status != RTEMS_SUCCESSFUL)
263                 {
264                   rtems_semaphore_obtain (results_sem,
265                                           RTEMS_WAIT,
266                                           RTEMS_NO_TIMEOUT);
267                   error = RTEMS_TASK_CREATE_ERROR;
268                   show_test_results (error, return_status);
269                   exit (-1);
270                 }
271
272                                                 return_status  =  rtems_task_start
(producers_array[index][system_index],
273                                                    producer_task,
274                                                    system_index);
275               if (return_status != RTEMS_SUCCESSFUL)
276                 {
277                   rtems_semaphore_obtain (results_sem,
278                                           RTEMS_WAIT,
279                                           RTEMS_NO_TIMEOUT);
```

```
280                   error = RTEMS_TASK_START_ERROR;
281                   show_test_results (error, return_status);
282                   exit (-1);
283                 }
284               created_producers++;
285             }
286           if (index < NUMBER_OF_CONSUMERS)
287             {
288               /* Create a consumer */
289               consumer_task_name = index + 1;
290               return_status = rtems_task_create (consumer_task_name,
291                                                  CONSUMERS_PRIORITY,
292                                                  CONSUMERS_TASK_STACK_SIZE,
293                                                  CONSUMERS_TASK_MODE,
294                                                  CONSUMERS_TASK_ATTR,
295
&consumers_array[index][system_index]);
296               if (return_status != RTEMS_SUCCESSFUL)
297                 {
298                   rtems_semaphore_obtain (results_sem,
299                                           RTEMS_WAIT,
300                                           RTEMS_NO_TIMEOUT);
301                   error = RTEMS_TASK_CREATE_ERROR;
302                   show_test_results (error, return_status);
303                   exit (-1);
304                 }
305
306                                                 return_status   =   rtems_task_start
(consumers_array[index][system_index],
307                                                    consumer_task,
308                                                    system_index);
309               if (return_status != RTEMS_SUCCESSFUL)
310                 {
311                   rtems_semaphore_obtain (results_sem,
312                                           RTEMS_WAIT,
313                                           RTEMS_NO_TIMEOUT);
314                   error = RTEMS_TASK_START_ERROR;
315                   show_test_results (error, return_status);
316                   exit (-1);
317                 }
318               created_consumers++;
319             }
320         }
321     }
322
323   /* Wait for all tasks or for an error. */
324   return_status = rtems_task_wake_after (TEST_TIMEOUT);
325
326   if (return_status != RTEMS_SUCCESSFUL)
327     {
328       error = RTEMS_TASK_WAKE_AFTER_ERROR;
329     }
330
331   rtems_semaphore_obtain (results_sem,
332                           RTEMS_WAIT,
333                           RTEMS_NO_TIMEOUT);
334   show_test_results (error, return_status);
335
```

```
336    exit (0);
337  }
338
339  void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
340  {
341    printf ("================================================\n");
342    printf ("Test Paramenters\n");
343    printf ("================================================\n");
344    printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
345    printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
346    printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
347    printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
348    printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
349    printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
350    printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
351    printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
352    printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
353    printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
354    printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
355    printf ("================================================\n");
356
357    if (error == NO_ERROR)
358      {
359        printf ("Systems created sucessfully:\n");
360      }
361    else
362      {
363        parse_error (error, return_status);
364      }
365    analyse_tasks_results ();
366  }
367
368  void analyse_tasks_results ()
369  {
370    int system_index, index;
371    rtems_unsigned32 running_consumers = 0;
372    rtems_unsigned32 running_producers = 0;
373    rtems_unsigned32 failed_consumers = 0;
374    rtems_unsigned32 failed_producers = 0;
375    int failed_consumer_index = -1;
376    int failed_consumer_system = -1;
377    int failed_producer_index = -1;
378    int failed_producer_system = -1;
379
380    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
381      {
382        /* Analyse producers array */
383         for (index = 0; (index < NUMBER_OF_PRODUCERS) && (((system_index + 1) *
(index + 1) + running_producers) < creat
ed_producers); index++)
384          {
385            if (producers_array[index][system_index] > 0)
386              {
387                running_producers++;
388              }
389            else if (producers_array[index][system_index] < 0)
390              {
391                failed_producers++;
```

```
392               if (failed_producer_index == -1)
393                 {
394                   failed_producer_index = index;
395                   failed_producer_system = system_index;
396                 }
397             }
398         }
399       /* Analyse consumers array */
400       for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
401         {
402           if (consumers_array[index][system_index] > 0)
403             {
404               running_consumers++;
405             }
406           else if (consumers_array[index][system_index] < 0)
407             {
408               failed_consumers++;
409               if (failed_consumer_index == -1)
410                 {
411                   failed_consumer_index = index;
412                   failed_consumer_system = system_index;
413                 }
414             }
415         }
416     }
417         printf ("Total number of successfully created producers: %d\n",
created_producers);
418    printf ("Running producers: %d\n", running_producers);
419     printf ("Producers that terminated successfuly: %d\n", (created_producers -
running_producers));
420    printf ("Number of failed producers: %d\n", failed_producers);
421    if (failed_producers > 0)
422      {
423             printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
424              printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
425      }
426         printf ("Total number of successfully created consumers: %d\n",
running_consumers);
427    printf ("Number of failed consumers: %d\n", failed_consumers);
428    if (failed_consumers > 0)
429      {
430             printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
431              printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
432      }
433  }
434
435  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
436  {
437       switch (error)
438         {
439         case RTEMS_SEMAPHORE_CREATE_ERROR:
440           printf ("Error creating semaphore.\n");
441           break;
442         case RTEMS_SEMAPHORE_RELEASE_ERROR:
```

```
443             printf ("Error signaling a semaphore.\n");
444             break;
445           case RTEMS_SEMAPHORE_OBTAIN_ERROR:
446             printf ("Error obtaining a semaphore.\n");
447             break;
448           case RTEMS_TASK_CREATE_ERROR:
449             printf ("Error creating task.\n");
450             break;
451           case RTEMS_TASK_START_ERROR:
452             printf ("Error starting task.\n");
453             break;
454           case RTEMS_TASK_WAKE_AFTER_ERROR:
455             printf ("Error waiting for tasks to finish.\n");
456             break;
457           case RTEMS_TASK_IDENT_ERROR:
458             printf ("Error obtaining thread ID.\n");
459             break;
460           default:
461             printf ("Unknown failure. Possible bug in workload.\n");
462             break;
463           }
464         show_error_code (return_status);
465   }
466
467   void show_error_code (rtems_status_code return_status)
468   {
469     printf ("Error code: ");
470     switch (return_status)
471       {
472       case RTEMS_TASK_EXITTED:
473         printf ("RTEMS_TASK_EXITTED");
474         break;
475       case RTEMS_MP_NOT_CONFIGURED:
476         printf ("RTEMS_MP_NOT_CONFIGURED");
477         break;
478       case RTEMS_INVALID_NAME:
479         printf ("RTEMS_INVALID_NAME");
480         break;
481       case RTEMS_INVALID_ID:
482         printf ("RTEMS_INVALID_ID");
483         break;
484       case RTEMS_TOO_MANY:
485         printf ("RTEMS_TOO_MANY");
486         break;
487       case RTEMS_TIMEOUT:
488         printf ("RTEMS_TIMEOUT");
489         break;
490       case RTEMS_OBJECT_WAS_DELETED:
491         printf ("RTEMS_OBJECT_WAS_DELETED");
492         break;
493       case RTEMS_INVALID_SIZE:
494         printf ("RTEMS_INVALID_SIZE");
495         break;
496       case RTEMS_INVALID_ADDRESS:
497         printf ("RTEMS_INVALID_ADDRESS");
498         break;
499       case RTEMS_INVALID_NUMBER:
500         printf ("RTEMS_INVALID_NUMBER");
```

```
501       break;
502     case RTEMS_NOT_DEFINED:
503       printf ("RTEMS_NOT_DEFINED");
504       break;
505     case RTEMS_RESOURCE_IN_USE:
506       printf ("RTEMS_RESOURCE_IN_USE");
507       break;
508     case RTEMS_UNSATISFIED:
509       printf ("RTEMS_UNSATISFIED");
510       break;
511     case RTEMS_INCORRECT_STATE:
512       printf ("RTEMS_INCORRECT_STATE");
513       break;
514     case RTEMS_ALREADY_SUSPENDED:
515       printf ("RTEMS_ALREADY_SUSPENDED");
516       break;
517     case RTEMS_ILLEGAL_ON_SELF:
518       printf ("RTEMS_ILLEGAL_ON_SELF");
519       break;
520     case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
521       printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
522       break;
523     case RTEMS_CALLED_FROM_ISR:
524       printf ("RTEMS_CALLED_FROM_ISR");
525       break;
526     case RTEMS_INVALID_PRIORITY:
527       printf ("RTEMS_INVALID_PRIORITY");
528       break;
529     case RTEMS_INVALID_CLOCK:
530       printf ("RTEMS_INVALID_CLOCK");
531       break;
532     case RTEMS_INVALID_NODE:
533       printf ("RTEMS_INVALID_NODE");
534       break;
535     case RTEMS_NOT_CONFIGURED:
536       printf ("RTEMS_NOT_CONFIGURED");
537       break;
538     case RTEMS_NOT_OWNER_OF_RESOURCE:
539       printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
540       break;
541     case RTEMS_NOT_IMPLEMENTED:
542       printf ("RTEMS_NOT_IMPLEMENTED");
543       break;
544     case RTEMS_INTERNAL_ERROR:
545       printf ("RTEMS_INTERNAL_ERROR");
546       break;
547     case RTEMS_NO_MEMORY:
548       printf ("RTEMS_NO_MEMORY");
549       break;
550     case RTEMS_IO_ERROR:
551       printf ("RTEMS_IO_ERROR");
552       break;
553     case RTEMS_PROXY_BLOCKING:
554       printf ("RTEMS_PROXY_BLOCKING");
555       break;
556     default:
557       printf ("UNKNOWN ERROR CODE");
558     }
```

```
559    printf ("\n");
560  }
561
562  /* configuration information */
563
564  #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
565  #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
566
567  #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
568   #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
569
570  #define CONFIGURE_MAXIMUM_SEMAPHORES (NUMBER_OF_SYSTEMS + 1)
571
572  #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
573
574  #define CONFIGURE_INIT
575
576  #include <confdefs.h>
577
578  /* end of file */
```

smp-producer.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Semaphore Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-smp/smp-producer.c,v $
 6   $Id: smp-producer.c,v 1.3 2003/10/24 13:23:02 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Semaphore
20   Manager. This file contains the producer task for this workload.
21
22   CREATED ON : 30-09-2003
23   CHANGED ON : $Date: 2003/10/24 13:23:02 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.3 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
```

```
33
34   HISTORY
35   $Log: smp-producer.c,v $
36   Revision 1.3  2003/10/24 13:23:02  lhenriques
37   Added function to set the error in the producers array.
38
39   Revision 1.2  2003/10/01 17:08:16  lhenriques
40   Added a Fatal Error user extension handler and the headings to the header file.
41
42   Revision 1.1  2003/10/01 14:56:39  lhenriques
43   First version of the stress-testing workload for the semaphore manager.
44
45   ***************************************************************************/
46
47 #include <bsp.h>
48 #include <stdio.h>
49 #include "rtems-cmp-cl-smp.h"
50
51 void set_producer_result (int res, rtems_id task_id, rtems_unsigned32 system)
52 {
53   int i;
54   for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
55     {
56       if (producers_array[i][system] == task_id)
57         {
58           producers_array[i][system] = res;
59           return;
60         }
61     }
62   printf ("[ERROR] Could not obtain the producer %d index in producers
63   array.\n", task_id);
64 }
65
66 /*
67  * This task will execute a loop where the system resource (a semaphore) will be
68  * signalled several times (NUMBER_OF_PRODUCED_ITEMS). Between every signal, the
69  * task sleep for DELAY_BETWEEN_ITEMS time.
70  * If an error occurs, the task tries to obtains the results_sem, shows the test
71  * results and exists.
72  */
73 rtems_task producer_task (rtems_task_argument producer_arg)
74 {
75   /* Get system number index the semaphores array */
76   rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
77   /* Number of rtems_semaphore_release calls */
78   rtems_unsigned32 count = NUMBER_OF_PRODUCED_ITEMS;
79   /* Get the semaphore */
80   rtems_id sem_id = semaphores_array[system];
81   rtems_status_code return_status = RTEMS_NOT_DEFINED;
82   rtems_unsigned32 error = NO_ERROR;
83   rtems_id task_id;
84
85   return_status = rtems_task_ident (RTEMS_SELF,
86                                     RTEMS_SEARCH_ALL_NODES,
87                                     &task_id);
88   if (return_status != RTEMS_SUCCESSFUL)
89     {
90       rtems_semaphore_obtain (results_sem,
```

```
 91                                RTEMS_WAIT,
 92                                RTEMS_NO_TIMEOUT);
 93        error = RTEMS_TASK_IDENT_ERROR;
 94        show_test_results (error, return_status);
 95        analyse_tasks_results ();
 96        exit (-1);
 97      }
 98
 99    while (count-- > 0)
100      {
101        return_status = rtems_semaphore_release (sem_id);
102        if (return_status != RTEMS_SUCCESSFUL)
103          {
104            rtems_semaphore_obtain (results_sem,
105                                    RTEMS_WAIT,
106                                    RTEMS_NO_TIMEOUT);
107            set_producer_result (-1, task_id, system);
108            error = RTEMS_SEMAPHORE_RELEASE_ERROR;
109            show_test_results (error, return_status);
110            exit (-1);
111          }
112        /* Wait before signaling again the semaphore */
113        return_status = rtems_task_wake_after (DELAY_BETWEEN_ITEMS);
114        if (return_status != RTEMS_SUCCESSFUL)
115          {
116            rtems_semaphore_obtain (results_sem,
117                                    RTEMS_WAIT,
118                                    RTEMS_NO_TIMEOUT);
119            set_producer_result (-1, task_id, system);
120            error = RTEMS_TASK_WAKE_AFTER_ERROR;
121            show_test_results (error, return_status);
122            exit (-1);
123          }
124      }
125    set_producer_result (0, task_id, system);
126  }
```

smp-consumer.c

```
  1  /*****************************************************************************
  2   SRC-MODULE : Semaphore Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-smp/smp-consumer.c,v $
  6   $Id: smp-consumer.c,v 1.3 2003/10/24 13:22:40 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
```

```
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Semaphore
20   Manager. This file contains the consumer task for this workload.
21
22   CREATED ON : 30-09-2003
23   CHANGED ON : $Date: 2003/10/24 13:22:40 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.3 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: smp-consumer.c,v $
36   Revision 1.3  2003/10/24 13:22:40  lhenriques
37   Added function to set the error in the consumers array.
38
39   Revision 1.2  2003/10/01 17:08:16  lhenriques
40   Added a Fatal Error user extension handler and the headings to the header file.
41
42   Revision 1.1  2003/10/01 14:56:39  lhenriques
43   First version of the stress-testing workload for the semaphore manager.
44
45   ***************************************************************************/
46
47 #include <bsp.h>
48 #include <stdio.h>
49 #include "rtems-cmp-cl-smp.h"
50
51 void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
52 {
53   int i;
54   for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
55     {
56       if (consumers_array[i][system] == task_id)
57         {
58           consumers_array[i][system] = -1;
59           return;
60         }
61     }
62   printf ("[ERROR] Could not obtain the consumer %d index in consumers
63   array.\n", task_id);
64 }
65
66 /*
67  * This task will wait forever in the resource (semaphore). It exits only if an
68  * error occurs in the return status of the wait directive.
69  */
70 rtems_task consumer_task (rtems_task_argument consumer_arg)
71 {
72   /* Get system number index the semaphores array */
73   rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
74   /* Get the semaphore */
75   rtems_id sem_id = semaphores_array[system];
```

```
 76    rtems_status_code return_status = RTEMS_NOT_DEFINED;
 77    rtems_unsigned32 error = NO_ERROR;
 78    rtems_id task_id;
 79
 80    return_status = rtems_task_ident (RTEMS_SELF,
 81                                      RTEMS_SEARCH_ALL_NODES,
 82                                      &task_id);
 83    if (return_status != RTEMS_SUCCESSFUL)
 84      {
 85        rtems_semaphore_obtain (results_sem,
 86                                RTEMS_WAIT,
 87                                RTEMS_NO_TIMEOUT);
 88        error = RTEMS_TASK_IDENT_ERROR;
 89        show_test_results (error, return_status);
 90        exit (-1);
 91      }
 92    while (TRUE)
 93      {
 94        return_status = rtems_semaphore_obtain (sem_id,
 95                                                RTEMS_WAIT,
 96                                                RTEMS_NO_TIMEOUT);
 97        if (return_status != RTEMS_SUCCESSFUL)
 98          {
 99            rtems_semaphore_obtain (results_sem,
100                                    RTEMS_WAIT,
101                                    RTEMS_NO_TIMEOUT);
102            set_consumer_error (task_id, system);
103            error = RTEMS_SEMAPHORE_OBTAIN_ERROR;
104            show_test_results (error, return_status);
105            exit (-1);
106          }
107      }
108  }
```

B.3. Message Manager Workload

rtems-cmp-cl-msg.h

```
  1  /**************************************************************************
  2   SRC-MODULE : Semaphore Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-msg/rtems-cmp-cl-msg.h,v $
  6   $Id: rtems-cmp-cl-msg.h,v 1.2 2003/10/24 13:19:11 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
```

```
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Semaphore Manager.
20
21   CREATED ON : 02-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:19:11 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-msg.h,v $
35   Revision 1.2  2003/10/24 13:19:11  lhenriques
36   Changed default parameters.
37
38   Revision 1.1  2003/10/10 15:43:37  lhenriques
39   First version of the stress-testing workload for the message manager.
40
41
42   ***************************************************************************/
43
44 /*
45    * The following definitions are the parameters that shall be changed to
generated
46   * diferent loads on the target.
47   */
48
49 /* Number of producer tasks */
50 #define NUMBER_OF_PRODUCERS 8
51
52 /* Stack size for the producers tasks */
53 #define PRODUCERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
54
55 /* Producers priority */
56 #define PRODUCERS_PRIORITY 2
57
58 /* Producers tasks mode */
59 #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
60
61 /* Producers tasks attributes */
62 #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
63
64 /* Number of consumers tasks */
65 #define NUMBER_OF_CONSUMERS 8
66
67 /* Stack size for the consumers tasks */
68 #define CONSUMERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
69
70 /* Consumers tasks mode */
71 #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
72
73 /* Consumers tasks attributes */
74 #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
```

```
 75
 76  /* Consumers priority */
 77  #define CONSUMERS_PRIORITY 3
 78
 79  /* Number of systems (producers/consumers systems) */
 80  #define NUMBER_OF_SYSTEMS 2
 81
 82  /* Number of ticks to wait until workload finishes */
 83  #define TEST_TIMEOUT 20000
 84
 85  /*************************************************************************/
 86  /* The following definitions are workload dependent.                     */
 87  /*************************************************************************/
 88
 89  /* Number of messagens to send by each producer */
 90  #define MAX_MESSAGES 13
 91
 92  /* Messages maximum size */
 93  #define MAX_MESSAGE_SIZE 32
 94
 95  #define MESSAGE_QUEUE_ATTRIBUTES (RTEMS_FIFO | RTEMS_LOCAL)
 96
 97  /* Time to wait before producing another item */
 98  #define DELAY_BETWEEN_ITEMS 1
 99
100  /*
101   * Enumeration with all the possible errors.
102   */
103  enum {
104    NO_ERROR,
105    RTEMS_SEMAPHORE_CREATE_ERROR,
106    RTEMS_MESSAGE_QUEUE_CREATE_ERROR,
107    RTEMS_SEMAPHORE_OBTAIN_ERROR,
108    RTEMS_TASK_CREATE_ERROR,
109    RTEMS_TASK_START_ERROR,
110    RTEMS_TASK_WAKE_AFTER_ERROR,
111    RTEMS_TASK_IDENT_ERROR,
112    RTEMS_MESSAGE_QUEUE_SEND_ERROR,
113    RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR,
114    RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR
115  };
116
117  /* Arrays to store the producers and consumers IDs */
118  extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
119  extern rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
120
121  /* Array containing the message queues for all systems */
122  extern rtems_id msg_queues_array[NUMBER_OF_SYSTEMS];
123
124  /* Semaphore that shall be obtained in order to exit the workload */
125  extern rtems_id results_sem;
126
127  /*
128   * These are the producer and consumer tasks.
129   */
130  rtems_task producer_task (rtems_task_argument producer_arg);
131  rtems_task consumer_task (rtems_task_argument consumer_arg);
132
```

```
133  /*
134   * This function shall analyse the results in each task,
135   */
136  void analyse_tasks_results ();
137
138  /*
139   * This function simply checks the errors in the test
140  H */
141       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
142
143  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
144
145  /*
146   * This function switchs the return status parameter and prints a string with
147   * the corresponding error code.
148   */
149  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-msg.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Message Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-msg/rtems-cmp-cl-msg.c,v $
 6   $Id: rtems-cmp-cl-msg.c,v 1.2 2003/10/24 13:19:01 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Message Manager.
20
21   CREATED ON : 01-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:19:01 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-msg.c,v $
35   Revision 1.2  2003/10/24 13:19:01  lhenriques
```

```
36   Corrected some problems in the tasks analysis.
37
38   Revision 1.1  2003/10/10 15:43:37  lhenriques
39   First version of the stress-testing workload for the message manager.
40
41
42   ***************************************************************************/
43
44  #include <bsp.h>
45  #include <stdio.h>
46  #include "rtems-cmp-cl-msg.h"
47
48  rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
49  rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
50
51  rtems_id msg_queues_array[NUMBER_OF_SYSTEMS];
52
53  rtems_id results_sem;
54
55  int created_producers = 0;
56  int created_consumers = 0;
57
58  /*
59   * Fatal error handler. It is used to find out whether an halt has
60   * occurred during the workload execution.
61   */
62  rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
63                                       boolean is_internal,
64                                       rtems_unsigned32 the_error)
65  {
66    printf ("A Fatal error has occurred!\n");
67    printf ("Source: ");
68    /* Find source */
69    switch (the_source)
70      {
71      case INTERNAL_ERROR_CORE:
72        printf ("INTERNAL_ERROR_CORE");
73        break;
74      case INTERNAL_ERROR_RTEMS_API:
75        printf ("INTERNAL_ERROR_RTEMS_API");
76        break;
77      case INTERNAL_ERROR_POSIX_API:
78        printf ("INTERNAL_ERROR_POSIX_API");
79        break;
80      case INTERNAL_ERROR_ITRON_API:
81        printf ("INTERNAL_ERROR_ITRON_API");
82      default:
83        printf ("UNKOWN (%d)", the_source);
84      }
85    printf ("\n");
86
87    if (is_internal == TRUE)
88      {
89        printf ("It is an INTERNAL error.\n");
90
91        printf ("Error: ");
92        /* Find the error itself */
93        switch (the_error)
```

```
 94         {
 95         case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
 96           printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
 97           break;
 98         case INTERNAL_ERROR_NO_CPU_TABLE:
 99           printf ("INTERNAL_ERROR_NO_CPU_TABLE");
100           break;
101         case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
102           printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
103           break;
104         case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
105           printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
106           break;
107         case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
108           printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
109           break;
110         case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
111           printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
112           break;
113         case INTERNAL_ERROR_THREAD_EXITTED:
114           printf ("INTERNAL_ERROR_THREAD_EXITTED");
115           break;
116         case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
117           printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
118           break;
119         case INTERNAL_ERROR_INVALID_NODE:
120           printf ("INTERNAL_ERROR_INVALID_NODE");
121           break;
122         case INTERNAL_ERROR_NO_MPCI:
123           printf ("INTERNAL_ERROR_NO_MPCI");
124           break;
125         case INTERNAL_ERROR_BAD_PACKET:
126           printf ("INTERNAL_ERROR_BAD_PACKET");
127           break;
128         case INTERNAL_ERROR_OUT_OF_PACKETS:
129           printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
130           break;
131         case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
132           printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
133           break;
134         case INTERNAL_ERROR_OUT_OF_PROXIES:
135           printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
136           break;
137         case INTERNAL_ERROR_INVALID_GLOBAL_ID:
138           printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
139           break;
140         case INTERNAL_ERROR_BAD_STACK_HOOK:
141           printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
142           break;
143         case INTERNAL_ERROR_BAD_ATTRIBUTES:
144           printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
145           break;
146         default:
147           printf ("UNKNOWN (%d)", the_error);
148         }
149       printf ("\n");
150
151     }
```

```
152    else
153      {
154        printf ("It is NOT an internal error.\n");
155        /* Assume an RTEMS Classic API error... */
156        show_error_code (the_error);
157      }
158    analyse_tasks_results ();
159    exit (-1);
160  }
161
162  /*
163   * This structure defines the user extensions entry points
164   */
165  rtems_extensions_table user_extensions =
166    {
167      NULL,               /* task creation extension */
168      NULL,               /* task start extension */
169      NULL,               /* task restart extension */
170      NULL,               /* task delete extension */
171      NULL,               /* task switch extension */
172      NULL,               /* task begin extension */
173      NULL,               /* task exitted extension */
174      fatal_error_handler /* fatal error extension */
175    };
176
177
178  /*
179   * Init task is the first to be executed.
180   */
181  rtems_task Init(rtems_task_argument ignored)
182  {
183    rtems_unsigned32 error = NO_ERROR;
184    rtems_status_code return_status = RTEMS_NOT_DEFINED;
185    rtems_unsigned32 system_index;
186    rtems_unsigned32 index;
187    rtems_unsigned32 max_index;
188    rtems_name results_sem_name;
189    rtems_name table_name;
190    rtems_id table_id;
191
192    /* Add user extension table to handle fata error */
193    table_name = rtems_build_name ('U', 'S', 'E', 'R');
194    return_status = rtems_extension_create (table_name,
195                                            &user_extensions,
196                                            &table_id);
197
198    if (return_status != RTEMS_SUCCESSFUL)
199      {
200              printf  ("Error  registering  an  user  handler  to  the  fatal  error
extension.\n");
201        show_error_code (return_status);
202        exit (-1);
203      }
204
205    /* Create the results semaphore */
206    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
207    return_status = rtems_semaphore_create (results_sem_name,
208                                            1, /* Only one task can exit */
```

```
209                                          RTEMS_DEFAULT_ATTRIBUTES,
210                                          RTEMS_NO_PRIORITY,
211                                          &results_sem);
212    /* If this fails... exit */
213    if (return_status != RTEMS_SUCCESSFUL)
214      {
215        printf ("Error creating semaphore for exit point.\n");
216        show_error_code (return_status);
217        exit (-1);
218      }
219
220    /* Create all the NUMBER_OF_SYSTEMS systems. */
221    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
222      {
223        rtems_name producer_task_name;
224        rtems_name consumer_task_name;
225        rtems_name msg_queue_name;
226
227        /* Create the resource -- a message queue for each system */
228        msg_queue_name = system_index + 1;
229        return_status = rtems_message_queue_create (msg_queue_name,
230                                                                    (MAX_MESSAGES *
NUMBER_OF_PRODUCERS),
231                                                    MAX_MESSAGE_SIZE,
232                                                    MESSAGE_QUEUE_ATTRIBUTES,
233
&msg_queues_array[system_index]);
234
235        if (return_status != RTEMS_SUCCESSFUL)
236          {
237            rtems_semaphore_obtain (results_sem,
238                                    RTEMS_WAIT,
239                                    RTEMS_NO_TIMEOUT);
240            error = RTEMS_MESSAGE_QUEUE_CREATE_ERROR;
241            show_test_results (error, return_status);
242            analyse_tasks_results ();
243            exit (-1);
244          }
245
246        /* Create all the producers and consumers for this system */
247                  max_index  =  (NUMBER_OF_PRODUCERS  <=  NUMBER_OF_CONSUMERS  ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
248        for (index = 0; index < max_index; index++)
249          {
250            if (index < NUMBER_OF_PRODUCERS)
251              {
252                /* Create a producer */
253                producer_task_name = index + 1;
254                return_status = rtems_task_create (producer_task_name,
255                                                   PRODUCERS_PRIORITY,
256                                                   PRODUCERS_TASK_STACK_SIZE,
257                                                   PRODUCERS_TASK_MODE,
258                                                   PRODUCERS_TASK_ATTR,
259
&producers_array[index][system_index]);
260                if (return_status != RTEMS_SUCCESSFUL)
261                  {
262                    rtems_semaphore_obtain (results_sem,
```

```
263                                         RTEMS_WAIT,
264                                         RTEMS_NO_TIMEOUT);
265                   error = RTEMS_TASK_CREATE_ERROR;
266                   show_test_results (error, return_status);
267                   analyse_tasks_results ();
268                   exit (-1);
269                 }
270
271                                               return_status   =   rtems_task_start
(producers_array[index][system_index],
272                                                  producer_task,
273                                                  system_index);
274               if (return_status != RTEMS_SUCCESSFUL)
275                 {
276                   rtems_semaphore_obtain (results_sem,
277                                         RTEMS_WAIT,
278                                         RTEMS_NO_TIMEOUT);
279                   error = RTEMS_TASK_START_ERROR;
280                   show_test_results (error, return_status);
281                   analyse_tasks_results ();
282                   exit (-1);
283                 }
284               created_producers++;
285             }
286           if (index < NUMBER_OF_CONSUMERS)
287             {
288               /* Create a consumer */
289               consumer_task_name = index + 1;
290               return_status = rtems_task_create (consumer_task_name,
291                                                  CONSUMERS_PRIORITY,
292                                                  CONSUMERS_TASK_STACK_SIZE,
293                                                  CONSUMERS_TASK_MODE,
294                                                  CONSUMERS_TASK_ATTR,
295
&consumers_array[index][system_index]);
296               if (return_status != RTEMS_SUCCESSFUL)
297                 {
298                   rtems_semaphore_obtain (results_sem,
299                                         RTEMS_WAIT,
300                                         RTEMS_NO_TIMEOUT);
301                   error = RTEMS_TASK_CREATE_ERROR;
302                   show_test_results (error, return_status);
303                   analyse_tasks_results ();
304                   exit (-1);
305                 }
306
307                                               return_status   =   rtems_task_start
(consumers_array[index][system_index],
308                                                  consumer_task,
309                                                  system_index);
310               if (return_status != RTEMS_SUCCESSFUL)
311                 {
312                   rtems_semaphore_obtain (results_sem,
313                                         RTEMS_WAIT,
314                                         RTEMS_NO_TIMEOUT);
315                   error = RTEMS_TASK_START_ERROR;
316                   show_test_results (error, return_status);
317                   analyse_tasks_results ();
```

```
318                   exit (-1);
319                 }
320               created_consumers++;
321             }
322         }
323     }
324
325   /* Wait for all tasks or for an error. */
326   return_status = rtems_task_wake_after (TEST_TIMEOUT);
327
328   if (return_status != RTEMS_SUCCESSFUL)
329     {
330       error = RTEMS_TASK_WAKE_AFTER_ERROR;
331     }
332
333   rtems_semaphore_obtain (results_sem,
334                           RTEMS_WAIT,
335                           RTEMS_NO_TIMEOUT);
336   show_test_results (error, return_status);
337   analyse_tasks_results ();
338
339   exit (0);
340 }
341
342 void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
343 {
344   printf ("================================================\n");
345   printf ("Test Paramenters\n");
346   printf ("================================================\n");
347   printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
348   printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
349   printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
350   printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
351   printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
352   printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
353   printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
354   printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
355   printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
356   printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
357   printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
358   printf ("================================================\n");
359   /* Workload specific parameters */
360       printf ("Number of messages to produce: %d\n", (NUMBER_OF_SYSTEMS *
NUMBER_OF_PRODUCERS * MAX_MESSAGES));
361   printf ("Number of messages per producer: %d\n", MAX_MESSAGES);
362   printf ("Size of messages: %d\n", MAX_MESSAGE_SIZE);
363
364   if (error == NO_ERROR)
365     {
366       printf ("Systems created sucessfully:\n");
367     }
368   else
369     {
370       parse_error (error, return_status);
371     }
372 }
373
374 void analyse_tasks_results ()
```

```
375  {
376    int system_index, index;
377    int number_of_consumers = 0;
378    int number_of_producers = 0;
379    int failed_consumers = 0;
380    int failed_consumer_index = -1;
381    int failed_consumer_system = -1;
382    int failed_producers = 0;
383    int failed_producer_index = -1;
384    int failed_producer_system = -1;
385
386    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
387      {
388        /* Analyse producers array */
389        for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
390          {
391            if (producers_array[index][system_index] != 0)
392              {
393                number_of_producers++;
394                     if (rtems_get_class (producers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
395                  {
396                    failed_producers++;
397                    if (failed_producer_index == -1)
398                      {
399                        failed_producer_index = index;
400                        failed_producer_system = system_index;
401                      }
402                  }
403              }
404          }
405        /* Analyse consumers array */
406        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
407          {
408            if (consumers_array[index][system_index] != 0)
409              {
410                number_of_consumers++;
411                     if (rtems_get_class (consumers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
412                  {
413                    failed_consumers++;
414                    if (failed_consumer_index == -1)
415                      {
416                        failed_consumer_index = index;
417                        failed_consumer_system = system_index;
418                      }
419                  }
420              }
421          }
422      }
423          printf ("Total number of successfully created producers: %d\n",
number_of_producers);
424    printf ("Number of failed producers: %d\n", failed_producers);
425    if (failed_producers > 0)
426      {
427              printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
```

```
   428              printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
   429      }
   430          printf ("Total number of successfully created consumers: %d\n",
number_of_consumers);
   431    printf ("Number of failed consumers: %d\n", failed_consumers);
   432    if (failed_consumers > 0)
   433      {
   434              printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
   435              printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
   436      }
   437  }
   438
   439  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
   440  {
   441        switch (error)
   442          {
   443          case RTEMS_SEMAPHORE_CREATE_ERROR:
   444            printf ("Error creating semaphore.\n");
   445            break;
   446          case RTEMS_MESSAGE_QUEUE_CREATE_ERROR:
   447            printf ("Error creating message queue.\n");
   448            break;
   449          case RTEMS_SEMAPHORE_OBTAIN_ERROR:
   450            printf ("Error obtaining a semaphore.\n");
   451            break;
   452          case RTEMS_TASK_CREATE_ERROR:
   453            printf ("Error creating task.\n");
   454            break;
   455          case RTEMS_TASK_START_ERROR:
   456            printf ("Error starting task.\n");
   457            break;
   458          case RTEMS_TASK_WAKE_AFTER_ERROR:
   459            printf ("Error waiting for tasks to finish.\n");
   460            break;
   461          case RTEMS_TASK_IDENT_ERROR:
   462            printf ("Error obtaining thread ID.\n");
   463            break;
   464          case RTEMS_MESSAGE_QUEUE_SEND_ERROR:
   465            printf ("Error sending message.\n");
   466            break;
   467          case RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR:
   468            printf ("Error receiving message.\n");
   469            break;
   470          case RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR:
   471                    printf ("Error receiving message: size is different from
expected.\n");
   472            break;
   473          default:
   474            printf ("Unknown failure. Possible bug in workload.\n");
   475            break;
   476          }
   477        show_error_code (return_status);
   478  }
   479
   480  void show_error_code (rtems_status_code return_status)
```

```
481 {
482   printf ("Error code: ");
483   switch (return_status)
484     {
485     case RTEMS_TASK_EXITTED:
486       printf ("RTEMS_TASK_EXITTED");
487       break;
488     case RTEMS_MP_NOT_CONFIGURED:
489       printf ("RTEMS_MP_NOT_CONFIGURED");
490       break;
491     case RTEMS_INVALID_NAME:
492       printf ("RTEMS_INVALID_NAME");
493       break;
494     case RTEMS_INVALID_ID:
495       printf ("RTEMS_INVALID_ID");
496       break;
497     case RTEMS_TOO_MANY:
498       printf ("RTEMS_TOO_MANY");
499       break;
500     case RTEMS_TIMEOUT:
501       printf ("RTEMS_TIMEOUT");
502       break;
503     case RTEMS_OBJECT_WAS_DELETED:
504       printf ("RTEMS_OBJECT_WAS_DELETED");
505       break;
506     case RTEMS_INVALID_SIZE:
507       printf ("RTEMS_INVALID_SIZE");
508       break;
509     case RTEMS_INVALID_ADDRESS:
510       printf ("RTEMS_INVALID_ADDRESS");
511       break;
512     case RTEMS_INVALID_NUMBER:
513       printf ("RTEMS_INVALID_NUMBER");
514       break;
515     case RTEMS_NOT_DEFINED:
516       printf ("RTEMS_NOT_DEFINED");
517       break;
518     case RTEMS_RESOURCE_IN_USE:
519       printf ("RTEMS_RESOURCE_IN_USE");
520       break;
521     case RTEMS_UNSATISFIED:
522       printf ("RTEMS_UNSATISFIED");
523       break;
524     case RTEMS_INCORRECT_STATE:
525       printf ("RTEMS_INCORRECT_STATE");
526       break;
527     case RTEMS_ALREADY_SUSPENDED:
528       printf ("RTEMS_ALREADY_SUSPENDED");
529       break;
530     case RTEMS_ILLEGAL_ON_SELF:
531       printf ("RTEMS_ILLEGAL_ON_SELF");
532       break;
533     case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
534       printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
535       break;
536     case RTEMS_CALLED_FROM_ISR:
537       printf ("RTEMS_CALLED_FROM_ISR");
538       break;
```

```
539       case RTEMS_INVALID_PRIORITY:
540         printf ("RTEMS_INVALID_PRIORITY");
541         break;
542       case RTEMS_INVALID_CLOCK:
543         printf ("RTEMS_INVALID_CLOCK");
544         break;
545       case RTEMS_INVALID_NODE:
546         printf ("RTEMS_INVALID_NODE");
547         break;
548       case RTEMS_NOT_CONFIGURED:
549         printf ("RTEMS_NOT_CONFIGURED");
550         break;
551       case RTEMS_NOT_OWNER_OF_RESOURCE:
552         printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
553         break;
554       case RTEMS_NOT_IMPLEMENTED:
555         printf ("RTEMS_NOT_IMPLEMENTED");
556         break;
557       case RTEMS_INTERNAL_ERROR:
558         printf ("RTEMS_INTERNAL_ERROR");
559         break;
560       case RTEMS_NO_MEMORY:
561         printf ("RTEMS_NO_MEMORY");
562         break;
563       case RTEMS_IO_ERROR:
564         printf ("RTEMS_IO_ERROR");
565         break;
566       case RTEMS_PROXY_BLOCKING:
567         printf ("RTEMS_PROXY_BLOCKING");
568         break;
569       default:
570         printf ("UNKNOWN ERROR CODE");
571       }
572     printf ("\n");
573   }
574
575   /* configuration information */
576
577   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
578   #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
579
580   #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
581   #define CONFIGURE_INIT_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE)
582     #define  CONFIGURE_MAXIMUM_TASKS  (NUMBER_OF_SYSTEMS  *  (NUMBER_OF_PRODUCERS  +
NUMBER_OF_CONSUMERS) + 1)
583
584   #define CONFIGURE_MAXIMUM_SEMAPHORES 1
585
586   #define CONFIGURE_MAXIMUM_MESSAGE_QUEUES NUMBER_OF_SYSTEMS
587
588   #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
589
590   #define CONFIGURE_INIT
591
592   #include <confdefs.h>
593
594   /* end of file */
```

msg-producer.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Semaphore Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-msg/msg-producer.c,v $
 6   $Id: msg-producer.c,v 1.1 2003/10/10 15:43:37 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Message
20   Manager. This file contains the producer task for this workload.
21
22   CREATED ON : 02-10-2003
23   CHANGED ON : $Date: 2003/10/10 15:43:37 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: msg-producer.c,v $
36   Revision 1.1  2003/10/10 15:43:37  lhenriques
37   First version of the stress-testing workload for the message manager.
38
39
40   *****************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-msg.h"
45
46
47  void set_producer_error (rtems_id task_id, rtems_unsigned32 system)
48  {
49    int i;
50    for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
51      {
52        if (producers_array[i][system] == task_id)
53          {
54            producers_array[i][system] = -1;
```

```
     55           return;
     56         }
     57     }
     58      printf ("[ERROR] Could not obtain the producer %d index in producers
array.\n", task_id);
     59 }
     60
     61 /*
     62  * This task will execute a loop where the system resource (a message queue)
     63  * will be used several times (MAX_MESSAGES): a number of messages
     64  * (MAX_MESSAGES) with MAX_MESSAGE_SIZE size will be sent to the message queue,
     65  * with a time interval of DELAY_BETWEEN_ITEMS between each message.
     66  * If an error occurs, the task tries to obtains the results_sem, shows the test
     67  * results and exists the workload.
     68  */
     69 rtems_task producer_task (rtems_task_argument producer_arg)
     70 {
     71   /* Get system number index the message queues array */
     72   rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
     73   /* Number of rtems_message_queue_send calls */
     74   rtems_signed32 count = MAX_MESSAGES;
     75   /* Get the message queue */
     76   rtems_id msg_queue_id = msg_queues_array[system];
     77   char buffer [MAX_MESSAGE_SIZE];
     78   rtems_status_code return_status = RTEMS_NOT_DEFINED;
     79   rtems_unsigned32 error = NO_ERROR;
     80   rtems_id task_id;
     81
     82   return_status = rtems_task_ident (RTEMS_SELF,
     83                                     RTEMS_SEARCH_ALL_NODES,
     84                                     &task_id);
     85   if (return_status != RTEMS_SUCCESSFUL)
     86     {
     87       rtems_semaphore_obtain (results_sem,
     88                               RTEMS_WAIT,
     89                               RTEMS_NO_TIMEOUT);
     90       error = RTEMS_TASK_IDENT_ERROR;
     91       show_test_results (error, return_status);
     92       analyse_tasks_results ();
     93       exit (-1);
     94     }
     95
     96   while (count-- > 0)
     97     {
     98       return_status = rtems_message_queue_send (msg_queue_id,
     99                                                 buffer,
    100                                                 MAX_MESSAGE_SIZE);
    101       if (return_status != RTEMS_SUCCESSFUL)
    102         {
    103           rtems_semaphore_obtain (results_sem,
    104                                   RTEMS_WAIT,
    105                                   RTEMS_NO_TIMEOUT);
    106           set_producer_error (task_id, system);
    107           error = RTEMS_MESSAGE_QUEUE_SEND_ERROR;
    108           show_test_results (error, return_status);
    109           analyse_tasks_results ();
    110           exit (-1);
    111         }
```

```
112         /* Wait before sending other message */
113         return_status = rtems_task_wake_after (DELAY_BETWEEN_ITEMS);
114         if (return_status != RTEMS_SUCCESSFUL)
115           {
116             rtems_semaphore_obtain (results_sem,
117                                     RTEMS_WAIT,
118                                     RTEMS_NO_TIMEOUT);
119             set_producer_error (task_id, system);
120             error = RTEMS_TASK_WAKE_AFTER_ERROR;
121             show_test_results (error, return_status);
122             analyse_tasks_results ();
123             exit (-1);
124           }
125       }
126   }
```

msg-consumer.c

```
  1  /*****************************************************************************
  2   SRC-MODULE : Semaphore Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-msg/msg-consumer.c,v $
  6   $Id: msg-consumer.c,v 1.1 2003/10/10 15:43:37 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Message
 20   Manager. This file contains the consumer task for this workload.
 21
 22   CREATED ON : 02-10-2003
 23   CHANGED ON : $Date: 2003/10/10 15:43:37 $
 24   CHANGED BY : $Author: lhenriques $
 25
 26   $Revision: 1.1 $
 27   STICKY TAG : $Name:  $
 28
 29   INSPECTED ON:
 30   MODERATOR :
 31
 32   TABLES : none.
 33
 34   HISTORY
 35   $Log: msg-consumer.c,v $
 36   Revision 1.1  2003/10/10 15:43:37  lhenriques
 37   First version of the stress-testing workload for the message manager.
 38
```

```
39
40   ****************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-msg.h"
45
46  void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
47  {
48    int i;
49    for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
50      {
51        if (consumers_array[i][system] == task_id)
52          {
53            consumers_array[i][system] = -1;
54            return;
55          }
56      }
57       printf  ("[ERROR]  Could  not  obtain  the  consumer  %d  index  in  consumers
array.\n", task_id);
58  }
59
60  /*
61   * This task will wait forever in the resource (message queue) for messages
62   * comming from the producers. It exits only if an error occurs in the return
63   * status of the wait directive.
64   */
65  rtems_task consumer_task (rtems_task_argument consumer_arg)
66  {
67    /* Get system number index for the message queues array */
68    rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
69    /* Get the message queue */
70    rtems_id msg_queue_id = msg_queues_array[system];
71    char buffer[MAX_MESSAGE_SIZE];
72    rtems_unsigned32 msg_size;
73    rtems_status_code return_status = RTEMS_NOT_DEFINED;
74    rtems_unsigned32 error = NO_ERROR;
75    rtems_id task_id;
76
77    return_status = rtems_task_ident (RTEMS_SELF,
78                                      RTEMS_SEARCH_ALL_NODES,
79                                      &task_id);
80    if (return_status != RTEMS_SUCCESSFUL)
81      {
82        rtems_semaphore_obtain (results_sem,
83                                RTEMS_WAIT,
84                                RTEMS_NO_TIMEOUT);
85        error = RTEMS_TASK_IDENT_ERROR;
86        show_test_results (error, return_status);
87        analyse_tasks_results ();
88        exit (-1);
89      }
90    while (TRUE)
91      {
92        return_status = rtems_message_queue_receive (msg_queue_id,
93                                                     buffer,
94                                                     &msg_size,
95                                                     RTEMS_WAIT,
```

```
 96                                                      RTEMS_NO_TIMEOUT);
 97       if (return_status != RTEMS_SUCCESSFUL)
 98         {
 99           rtems_semaphore_obtain (results_sem,
100                                   RTEMS_WAIT,
101                                   RTEMS_NO_TIMEOUT);
102           error = RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR;
103           set_consumer_error (task_id, system);
104           show_test_results (error, return_status);
105           analyse_tasks_results ();
106           exit (-1);
107         }
108       if (msg_size != MAX_MESSAGE_SIZE)
109         {
110           rtems_semaphore_obtain (results_sem,
111                                   RTEMS_WAIT,
112                                   RTEMS_NO_TIMEOUT);
113           error = RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR;
114           set_consumer_error (task_id, system);
115           show_test_results (error, return_status);
116           analyse_tasks_results ();
117           exit (-1);
118         }
119     }
120 }
```

B.4. Signal Manager Workload

rtems-cmp-cl-sgn.h

```
 1  /*****************************************************************************
 2   SRC-MODULE : Signal Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-sgn/rtems-cmp-cl-sgn.h,v $
 6   $Id: rtems-cmp-cl-sgn.h,v 1.2 2003/10/24 13:20:36 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Signal Manager.
20
21   CREATED ON : 02-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:20:36 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
```

```
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-sgn.h,v $
35   Revision 1.2  2003/10/24 13:20:36  lhenriques
36   Changed default parameters.
37
38   Revision 1.1  2003/10/10 15:42:39  lhenriques
39   First version of the stress-testing workload for the signal manager.
40
41
42   ***************************************************************************/
43
44  /*
45     * The following definitions are the parameters that shall be changed to
generated
46   * diferent loads on the target.
47   */
48
49  /* Number of producer tasks */
50  #define NUMBER_OF_PRODUCERS 64
51
52  /* Stack size for the producers tasks */
53  #define PRODUCERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
54
55  /* Producers priority */
56  #define PRODUCERS_PRIORITY 3
57
58  /* Producers tasks mode */
59  #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
60
61  /* Producers tasks attributes */
62  #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
63
64  /* Number of consumers tasks */
65  #define NUMBER_OF_CONSUMERS 64
66
67  /* Stack size for the consumers tasks */
68  #define CONSUMERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
69
70  /* Consumers tasks mode */
71  #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
72
73  /* Consumers tasks attributes */
74  #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
75
76  /*
77   * Consumers priority
78   * NOTE: for this workload, it shall always be higher than the producer's
79   */
80  #define CONSUMERS_PRIORITY 2
81
82  /* Number of systems (producers/consumers systems) */
```

```
 83 #define NUMBER_OF_SYSTEMS 4
 84
 85 /* Number of ticks to wait until workload finishes */
 86 #define TEST_TIMEOUT 20000
 87
 88 /*****************************************************************************/
 89 /* The following definitions are workload dependent.                         */
 90 /*****************************************************************************/
 91
 92 /* Mode of the ASR */
 93 #define SIGNAL_HANDLER_MODE RTEMS_PREEMPT
 94
 95 /* Semaphores initial count */
 96 #define SEMAPHORES_INITIAL_COUNT 0
 97
 98 /* Semaphores attributes */
 99 #define SEMAPHORES_ATTRIBUTES RTEMS_DEFAULT_ATTRIBUTES
100
101 /* Semaphores priority */
102 #define SEMAPHORES_PRIORITY RTEMS_NO_PRIORITY
103
104 /* Number of signals each producer has to send */
105 #define NUMBER_OF_SIGNALS 320
106
107 /* Time to wait before producing another item */
108 #define DELAY_BETWEEN_ITEMS 1
109
110 /*
111  * Enumeration with all the possible errors.
112  */
113 enum {
114   NO_ERROR,
115   RTEMS_SEMAPHORE_CREATE_ERROR,
116   RTEMS_SEMAPHORE_OBTAIN_ERROR,
117   RTEMS_SEMAPHORE_RELEASE_ERROR,
118   RTEMS_SEMAPHORE_FLUSH_ERROR,
119   RTEMS_TASK_CREATE_ERROR,
120   RTEMS_TASK_START_ERROR,
121   RTEMS_TASK_WAKE_AFTER_ERROR,
122   RTEMS_TASK_IDENT_ERROR,
123   RTEMS_SIGNAL_CATCH_ERROR,
124   RTEMS_SIGNAL_SEND_ERROR,
125   RTEMS_INVALID_SIGNAL_ERROR,
126   RTEMS_GET_CONSUMER_INFO_ERROR
127 };
128
129 typedef struct {
130   rtems_id task_id;
131   rtems_id sem_id;
132   int signals_sent[32];
133   int signals_received[32];
134 } consumer_info;
135
136 /* Arrays to store the producers and consumers IDs */
137 extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
138
139 /* Array with consumers semaphores. */
140 extern consumer_info consumers_info[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
```

```
141
142  /* Semaphore that shall be obtained in order to exit the workload */
143  extern rtems_id results_sem;
144
145  /*
146   * These are the producer and consumer tasks.
147   */
148  rtems_task producer_task (rtems_task_argument producer_arg);
149  rtems_task consumer_task (rtems_task_argument consumer_arg);
150
151  /*
152   * This function shall analyse the results in each task,
153   */
154  void analyse_tasks_results ();
155
156  /*
157   * This function simply checks the errors in the test
158  H */
159       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
160
161  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
162
163  /*
164   * This function switchs the return status parameter and prints a string with
165   * the corresponding error code.
166   */
167  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-sgn.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Signal Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-sgn/rtems-cmp-cl-sgn.c,v $
 6   $Id: rtems-cmp-cl-sgn.c,v 1.2 2003/10/24 13:20:20 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Signal Manager.
20
21   CREATED ON : 01-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:20:20 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
```

```
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-sgn.c,v $
35   Revision 1.2  2003/10/24 13:20:20  lhenriques
36   Corrected some problems in the tasks analysis.
37
38   Revision 1.1  2003/10/10 15:42:39  lhenriques
39   First version of the stress-testing workload for the signal manager.
40
41
42   ***************************************************************************/
43
44  #include <bsp.h>
45  #include <stdio.h>
46  #include "rtems-cmp-cl-sgn.h"
47
48  rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
49
50  consumer_info consumers_info[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
51
52  rtems_id results_sem;
53
54  int created_producers = 0;
55  int created_consumers = 0;
56
57  /*
58   * Fatal error handler. It is used to find out whether an halt has
59   * occurred during the workload execution.
60   */
61  rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
62                                       boolean is_internal,
63                                       rtems_unsigned32 the_error)
64  {
65    printf ("A Fatal error has occurred!\n");
66    printf ("Source: ");
67    /* Find source */
68    switch (the_source)
69      {
70      case INTERNAL_ERROR_CORE:
71        printf ("INTERNAL_ERROR_CORE");
72        break;
73      case INTERNAL_ERROR_RTEMS_API:
74        printf ("INTERNAL_ERROR_RTEMS_API");
75        break;
76      case INTERNAL_ERROR_POSIX_API:
77        printf ("INTERNAL_ERROR_POSIX_API");
78        break;
79      case INTERNAL_ERROR_ITRON_API:
80        printf ("INTERNAL_ERROR_ITRON_API");
81      default:
82        printf ("UNKOWN (%d)", the_source);
83      }
```

```
 84    printf ("\n");
 85
 86    if (is_internal == TRUE)
 87      {
 88        printf ("It is an INTERNAL error.\n");
 89
 90        printf ("Error: ");
 91        /* Find the error itself */
 92        switch (the_error)
 93          {
 94          case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
 95            printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
 96            break;
 97          case INTERNAL_ERROR_NO_CPU_TABLE:
 98            printf ("INTERNAL_ERROR_NO_CPU_TABLE");
 99            break;
100          case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
101            printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
102            break;
103          case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
104            printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
105            break;
106          case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
107            printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
108            break;
109          case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
110            printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
111            break;
112          case INTERNAL_ERROR_THREAD_EXITTED:
113            printf ("INTERNAL_ERROR_THREAD_EXITTED");
114            break;
115          case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
116            printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
117            break;
118          case INTERNAL_ERROR_INVALID_NODE:
119            printf ("INTERNAL_ERROR_INVALID_NODE");
120            break;
121          case INTERNAL_ERROR_NO_MPCI:
122            printf ("INTERNAL_ERROR_NO_MPCI");
123            break;
124          case INTERNAL_ERROR_BAD_PACKET:
125            printf ("INTERNAL_ERROR_BAD_PACKET");
126            break;
127          case INTERNAL_ERROR_OUT_OF_PACKETS:
128            printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
129            break;
130          case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
131            printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
132            break;
133          case INTERNAL_ERROR_OUT_OF_PROXIES:
134            printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
135            break;
136          case INTERNAL_ERROR_INVALID_GLOBAL_ID:
137            printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
138            break;
139          case INTERNAL_ERROR_BAD_STACK_HOOK:
140            printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
141            break;
```

```
142           case INTERNAL_ERROR_BAD_ATTRIBUTES:
143             printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
144             break;
145           default:
146             printf ("UNKNOWN (%d)", the_error);
147           }
148         printf ("\n");
149
150       }
151     else
152       {
153         printf ("It is NOT an internal error.\n");
154         /* Assume an RTEMS Classic API error... */
155         show_error_code (the_error);
156       }
157     analyse_tasks_results ();
158     exit (-1);
159   }
160
161   /*
162    * This structure defines the user extensions entry points
163    */
164   rtems_extensions_table user_extensions =
165     {
166       NULL,               /* task creation extension */
167       NULL,               /* task start extension */
168       NULL,               /* task restart extension */
169       NULL,               /* task delete extension */
170       NULL,               /* task switch extension */
171       NULL,               /* task begin extension */
172       NULL,               /* task exitted extension */
173       fatal_error_handler /* fatal error extension */
174     };
175
176   /*
177    * Init task is the first to be executed.
178    */
179   rtems_task Init(rtems_task_argument ignored)
180   {
181     rtems_unsigned32 error = NO_ERROR;
182     rtems_status_code return_status = RTEMS_NOT_DEFINED;
183     rtems_unsigned32 system_index;
184     rtems_unsigned32 index;
185     rtems_unsigned32 max_index;
186     rtems_name results_sem_name;
187     rtems_name table_name;
188     rtems_id table_id;
189
190     /* Add user extension table to handle fata error */
191     table_name = rtems_build_name ('U', 'S', 'E', 'R');
192     return_status = rtems_extension_create (table_name,
193                                             &user_extensions,
194                                             &table_id);
195
196     if (return_status != RTEMS_SUCCESSFUL)
197       {
198               printf  ("Error  registering  an  user  handler  to  the  fatal  error
    extension.\n");
```

```
199         show_error_code (return_status);
200         exit (-1);
201       }
202
203     /* Create the results semaphore */
204     results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
205     return_status = rtems_semaphore_create (results_sem_name,
206                                             1, /* Only one task can exit */
207                                             RTEMS_DEFAULT_ATTRIBUTES,
208                                             RTEMS_NO_PRIORITY,
209                                             &results_sem);
210     /* If this fails... exit */
211     if (return_status != RTEMS_SUCCESSFUL)
212       {
213         printf ("Error creating semaphore for exit point.\n");
214         show_error_code (return_status);
215         exit (-1);
216       }
217
218     /* Create all the NUMBER_OF_SYSTEMS systems. */
219     for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
220       {
221         rtems_name producer_task_name;
222         rtems_name consumer_task_name;
223
224         /* Create all the producers and consumers for this system */
225                   max_index   =   (NUMBER_OF_PRODUCERS   <=   NUMBER_OF_CONSUMERS   ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
226         for (index = 0; index < max_index; index++)
227           {
228             if (index < NUMBER_OF_PRODUCERS)
229               {
230                 /* Create a producer */
231                 producer_task_name = index + 1;
232                 return_status = rtems_task_create (producer_task_name,
233                                                    PRODUCERS_PRIORITY,
234                                                    PRODUCERS_TASK_STACK_SIZE,
235                                                    PRODUCERS_TASK_MODE,
236                                                    PRODUCERS_TASK_ATTR,
237
&producers_array[index][system_index]);
238                 if (return_status != RTEMS_SUCCESSFUL)
239                   {
240                     rtems_semaphore_obtain (results_sem,
241                                             RTEMS_WAIT,
242                                             RTEMS_NO_TIMEOUT);
243                     error = RTEMS_TASK_CREATE_ERROR;
244                     show_test_results (error, return_status);
245                     analyse_tasks_results ();
246                     exit (-1);
247                   }
248               }
249             if (index < NUMBER_OF_CONSUMERS)
250               {
251                 rtems_name consumer_sem_name;
252                 /* Create the consumer semaphore */
253                       consumer_sem_name = rtems_build_name ('C', 'S', (char)(65 +
system_index), (char)(65 + index));
```

```
254
255               return_status = rtems_semaphore_create (consumer_sem_name,
256                                                       0,
257                                                       RTEMS_DEFAULT_ATTRIBUTES,
258                                                       RTEMS_NO_PRIORITY,
259
&((consumer_info)(consumers_info[index][system_index])).sem_id);
260               if (return_status != RTEMS_SUCCESSFUL)
261                 {
262                   rtems_semaphore_obtain (results_sem,
263                                           RTEMS_WAIT,
264                                           RTEMS_NO_TIMEOUT);
265                   error = RTEMS_SEMAPHORE_CREATE_ERROR;
266                   show_test_results (error, return_status);
267                   analyse_tasks_results ();
268                   exit (-1);
269                 }
270
271               /* Create a consumer */
272               consumer_task_name = index + 1;
273               return_status = rtems_task_create (consumer_task_name,
274                                                  CONSUMERS_PRIORITY,
275                                                  CONSUMERS_TASK_STACK_SIZE,
276                                                  CONSUMERS_TASK_MODE,
277                                                  CONSUMERS_TASK_ATTR,
278
&((consumer_info)(consumers_info[index][system_index])).task_id);
279               if (return_status != RTEMS_SUCCESSFUL)
280                 {
281                   rtems_semaphore_obtain (results_sem,
282                                           RTEMS_WAIT,
283                                           RTEMS_NO_TIMEOUT);
284                   error = RTEMS_TASK_CREATE_ERROR;
285                   show_test_results (error, return_status);
286                   analyse_tasks_results ();
287                   exit (-1);
288                 }
289             }
290         }
291       for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
292         {
293                                             return_status    =    rtems_task_start
(consumers_info[index][system_index].task_id,
294                                              consumer_task,
295                                              system_index);
296           if (return_status != RTEMS_SUCCESSFUL)
297             {
298               rtems_semaphore_obtain (results_sem,
299                                       RTEMS_WAIT,
300                                       RTEMS_NO_TIMEOUT);
301               error = RTEMS_TASK_START_ERROR;
302               show_test_results (error, return_status);
303               analyse_tasks_results ();
304               exit (-1);
305             }
306           created_consumers++;
307         }
308       for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
```

```
309          {
310                                                return_status    =    rtems_task_start
(producers_array[index][system_index],
311                                                 producer_task,
312                                                 system_index);
313            if (return_status != RTEMS_SUCCESSFUL)
314              {
315                rtems_semaphore_obtain (results_sem,
316                                        RTEMS_WAIT,
317                                        RTEMS_NO_TIMEOUT);
318                error = RTEMS_TASK_START_ERROR;
319                show_test_results (error, return_status);
320                analyse_tasks_results ();
321                exit (-1);
322              }
323            created_producers++;
324          }
325      }
326
327    /* Wait for all tasks or for an error. */
328    return_status = rtems_task_wake_after (TEST_TIMEOUT);
329
330    if (return_status != RTEMS_SUCCESSFUL)
331      {
332        error = RTEMS_TASK_WAKE_AFTER_ERROR;
333      }
334
335    rtems_semaphore_obtain (results_sem,
336                            RTEMS_WAIT,
337                            RTEMS_NO_TIMEOUT);
338    show_test_results (error, return_status);
339    analyse_tasks_results ();
340
341    exit (0);
342  }
343
344  void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
345  {
346    printf ("================================================\n");
347    printf ("Test Paramenters\n");
348    printf ("================================================\n");
349    printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
350    printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
351    printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
352    printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
353    printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
354    printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
355    printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
356    printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
357    printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
358    printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
359    printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
360    printf ("================================================\n");
361    /* Workload specific parameters */
362
363    if (error == NO_ERROR)
364      {
365        printf ("Systems created sucessfully:\n");
```

```
366       }
367     else
368       {
369         parse_error (error, return_status);
370       }
371   }
372
373   void analyse_tasks_results ()
374   {
375     int system_index, index, i;
376     int number_of_consumers = 0;
377     int number_of_producers = 0;
378     int failed_consumers = 0;
379     int failed_consumer_index = -1;
380     int failed_consumer_system = -1;
381     int failed_producers = 0;
382     int failed_producer_index = -1;
383     int failed_producer_system = -1;
384
385     for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
386       {
387         /* Analyse producers array */
388         for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
389           {
390             if (producers_array[index][system_index] != 0)
391               {
392                 number_of_producers++;
393                      if (rtems_get_class (producers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
394                   {
395                     failed_producers++;
396                     if (failed_producer_index == -1)
397                       {
398                         failed_producer_index = index;
399                         failed_producer_system = system_index;
400                       }
401                   }
402               }
403           }
404         /* Analyse consumers array */
405         for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
406           {
407             if (consumers_info[index][system_index].task_id != 0)
408               {
409                 number_of_consumers++;
410                   if (rtems_get_class (consumers_info[index][system_index].task_id)
!= OBJECTS_RTEMS_TASKS)
411                   {
412                     failed_consumers++;
413                     if (failed_consumer_index == -1)
414                       {
415                         failed_consumer_index = index;
416                         failed_consumer_system = system_index;
417                       }
418                   }
419               }
420           }
421       }
```

```
   422    /*printf ("Total number of signals: %d\n", sig_count);*/
   423         printf ("Total number of successfully created producers: %d\n",
number_of_producers);
   424    printf ("Number of failed producers: %d\n", failed_producers);
   425    if (failed_producers > 0)
   426      {
   427             printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
   428              printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
   429      }
   430         printf ("Total number of successfully created consumers: %d\n",
number_of_consumers);
   431    printf ("Number of failed consumers: %d\n", failed_consumers);
   432    if (failed_consumers > 0)
   433      {
   434             printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
   435              printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
   436      }
   437    /* Show number of signals received by each consumer */
   438    printf ("Analysing signals sent/received...\n");
   439    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
   440      {
   441        rtems_boolean system_ok = TRUE;
   442        printf ("System %d.............", system_index);
   443        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
   444          {
   445            for (i = 0; i < 32; i++)
   446              {
   447                if (consumers_info[index][system_index].signals_sent[i] !=
   448                    consumers_info[index][system_index].signals_received[i])
   449                  {
   450                     printf ("\nERROR: %d RTEMS_SIGNAL_%d signals sent to consumer
%d, only %d received.\n",
   451                             consumers_info[index][system_index].signals_sent[i],
   452                             i,
   453                             index,
   454
consumers_info[index][system_index].signals_received[i]);
   455                    system_ok = FALSE;
   456                  }
   457              }
   458          }
   459        if (system_ok == TRUE)
   460          {
   461            printf ("OK\n");
   462          }
   463      }
   464  }
   465
   466  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
   467  {
   468        switch (error)
   469          {
   470          case RTEMS_SEMAPHORE_CREATE_ERROR:
   471            printf ("Error creating semaphore.\n");
```

```
472           break;
473         case RTEMS_SEMAPHORE_OBTAIN_ERROR:
474           printf ("Error obtaining a semaphore.\n");
475           break;
476         case RTEMS_SEMAPHORE_RELEASE_ERROR:
477           printf ("Error signaling a semaphore.\n");
478           break;
479         case RTEMS_SEMAPHORE_FLUSH_ERROR:
480           printf ("Error flushing semaphore.\n");
481           break;
482         case RTEMS_TASK_CREATE_ERROR:
483           printf ("Error creating task.\n");
484           break;
485         case RTEMS_TASK_START_ERROR:
486           printf ("Error starting task.\n");
487           break;
488         case RTEMS_TASK_WAKE_AFTER_ERROR:
489           printf ("Error waiting for tasks to finish.\n");
490           break;
491         case RTEMS_TASK_IDENT_ERROR:
492           printf ("Error obtaining thread ID.\n");
493           break;
494         case RTEMS_SIGNAL_CATCH_ERROR:
495           printf ("Error registering ASR.\n");
496           break;
497         case RTEMS_SIGNAL_SEND_ERROR:
498           printf ("Error sending signal.\n");
499           break;
500         case RTEMS_GET_CONSUMER_INFO_ERROR:
501           printf ("Error obtaining the consumer information.\n");
502           break;
503         default:
504           printf ("Unknown failure. Possible bug in workload.\n");
505           break;
506         }
507       show_error_code (return_status);
508 }
509
510 void show_error_code (rtems_status_code return_status)
511 {
512   printf ("Error code: ");
513   switch (return_status)
514     {
515     case RTEMS_TASK_EXITTED:
516       printf ("RTEMS_TASK_EXITTED");
517       break;
518     case RTEMS_MP_NOT_CONFIGURED:
519       printf ("RTEMS_MP_NOT_CONFIGURED");
520       break;
521     case RTEMS_INVALID_NAME:
522       printf ("RTEMS_INVALID_NAME");
523       break;
524     case RTEMS_INVALID_ID:
525       printf ("RTEMS_INVALID_ID");
526       break;
527     case RTEMS_TOO_MANY:
528       printf ("RTEMS_TOO_MANY");
529       break;
```

```
530       case RTEMS_TIMEOUT:
531         printf ("RTEMS_TIMEOUT");
532         break;
533       case RTEMS_OBJECT_WAS_DELETED:
534         printf ("RTEMS_OBJECT_WAS_DELETED");
535         break;
536       case RTEMS_INVALID_SIZE:
537         printf ("RTEMS_INVALID_SIZE");
538         break;
539       case RTEMS_INVALID_ADDRESS:
540         printf ("RTEMS_INVALID_ADDRESS");
541         break;
542       case RTEMS_INVALID_NUMBER:
543         printf ("RTEMS_INVALID_NUMBER");
544         break;
545       case RTEMS_NOT_DEFINED:
546         printf ("RTEMS_NOT_DEFINED");
547         break;
548       case RTEMS_RESOURCE_IN_USE:
549         printf ("RTEMS_RESOURCE_IN_USE");
550         break;
551       case RTEMS_UNSATISFIED:
552         printf ("RTEMS_UNSATISFIED");
553         break;
554       case RTEMS_INCORRECT_STATE:
555         printf ("RTEMS_INCORRECT_STATE");
556         break;
557       case RTEMS_ALREADY_SUSPENDED:
558         printf ("RTEMS_ALREADY_SUSPENDED");
559         break;
560       case RTEMS_ILLEGAL_ON_SELF:
561         printf ("RTEMS_ILLEGAL_ON_SELF");
562         break;
563       case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
564         printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
565         break;
566       case RTEMS_CALLED_FROM_ISR:
567         printf ("RTEMS_CALLED_FROM_ISR");
568         break;
569       case RTEMS_INVALID_PRIORITY:
570         printf ("RTEMS_INVALID_PRIORITY");
571         break;
572       case RTEMS_INVALID_CLOCK:
573         printf ("RTEMS_INVALID_CLOCK");
574         break;
575       case RTEMS_INVALID_NODE:
576         printf ("RTEMS_INVALID_NODE");
577         break;
578       case RTEMS_NOT_CONFIGURED:
579         printf ("RTEMS_NOT_CONFIGURED");
580         break;
581       case RTEMS_NOT_OWNER_OF_RESOURCE:
582         printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
583         break;
584       case RTEMS_NOT_IMPLEMENTED:
585         printf ("RTEMS_NOT_IMPLEMENTED");
586         break;
587       case RTEMS_INTERNAL_ERROR:
```

```
588         printf ("RTEMS_INTERNAL_ERROR");
589         break;
590       case RTEMS_NO_MEMORY:
591         printf ("RTEMS_NO_MEMORY");
592         break;
593       case RTEMS_IO_ERROR:
594         printf ("RTEMS_IO_ERROR");
595         break;
596       case RTEMS_PROXY_BLOCKING:
597         printf ("RTEMS_PROXY_BLOCKING");
598         break;
599       default:
600         printf ("UNKNOWN ERROR CODE");
601       }
602     printf ("\n");
603   }
604
605   /* configuration information */
606
607   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
608   #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
609
610   #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
611   #define CONFIGURE_INIT_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
612   #define CONFIGURE_EXTRA_TASK_STACKS (1 * RTEMS_MINIMUM_STACK_SIZE)
613
614    #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
615
616   #define CONFIGURE_MAXIMUM_SEMAPHORES ((NUMBER_OF_SYSTEMS * NUMBER_OF_CONSUMERS)
+ 1)
617
618   #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
619
620   #define CONFIGURE_INIT
621
622   #include <confdefs.h>
623
624   /* end of file */
```

sgn-producer.c

```
1   /*****************************************************************************
2    SRC-MODULE : Signal Manager Workload
3    MODULE-VERS : N/A
4
5             $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-sgn/sgn-producer.c,v $
6    $Id: sgn-producer.c,v 1.2 2003/10/24 13:21:30 lhenriques Exp $
7    $State: Exp $
8    $Locker:  $
9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
```

```
15
16  AUTHOR : lhenriques
17
18  KEYWORDS : ----
19  PURPOSE : Stress the RTEMS Classical API for the Signal
20  Manager. This file contains the producer task for this workload.
21
22  CREATED ON : 02-10-2003
23  CHANGED ON : $Date: 2003/10/24 13:21:30 $
24  CHANGED BY : $Author: lhenriques $
25
26  $Revision: 1.2 $
27  STICKY TAG : $Name:  $
28
29  INSPECTED ON:
30  MODERATOR :
31
32  TABLES : none.
33
34  HISTORY
35  $Log: sgn-producer.c,v $
36  Revision 1.2  2003/10/24 13:21:30  lhenriques
37  Corrected a bug in the producer.
38
39  Revision 1.1  2003/10/10 15:42:39  lhenriques
40  First version of the stress-testing workload for the signal manager.
41
42
43  ******************************************************************************/
44
45 #include <bsp.h>
46 #include <stdio.h>
47 #include "rtems-cmp-cl-sgn.h"
48
49 void set_producer_error (rtems_id task_id, rtems_unsigned32 system)
50 {
51   int i;
52   for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
53     {
54       if (producers_array[i][system] == task_id)
55         {
56           producers_array[i][system] = -1;
57           return;
58         }
59     }
60      printf ("[ERROR] Could not obtain the producer %d index in producers
array.\n", task_id);
61 }
62
63 /*
64  * This task will execute a loop where the system resource (signal)
65  * will be used several times (NUMBER_OF_SIGNALS): signals will be sent to the
66   * message queue, with a time interval of DELAY_BETWEEN_ITEMS between each
message.
67  * If an error occurs, the task tries to obtains the results_sem, shows the test
68  * results and exists the workload.
69  */
70 rtems_task producer_task (rtems_task_argument producer_arg)
```

```
71  {
72    /* Get system number index */
73    rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
74    rtems_status_code return_status = RTEMS_NOT_DEFINED;
75    /* Number of signals to send */
76    rtems_signed32 count = NUMBER_OF_SIGNALS;
77    rtems_unsigned32 error = NO_ERROR;
78    rtems_id task_id;
79    rtems_id signal_task_id;
80    int consumers_index = 0;
81    int signal = 0;
82    rtems_signal_set signal_to_send = RTEMS_SIGNAL_0;
83    rtems_id sem_id;
84
85    return_status = rtems_task_ident (RTEMS_SELF,
86                                      RTEMS_SEARCH_ALL_NODES,
87                                      &task_id);
88    if (return_status != RTEMS_SUCCESSFUL)
89      {
90        rtems_semaphore_obtain (results_sem,
91                                RTEMS_WAIT,
92                                RTEMS_NO_TIMEOUT);
93        error = RTEMS_TASK_IDENT_ERROR;
94        show_test_results (error, return_status);
95        analyse_tasks_results ();
96        exit (-1);
97      }
98
99    while (count-- > 0)
100     {
101       signal_task_id = consumers_info [consumers_index][system].task_id;
102       sem_id = consumers_info [consumers_index][system].sem_id;
103
104       signal_to_send = 1 << signal;
105
106       return_status = rtems_signal_send(signal_task_id,
107                                         signal_to_send);
108
109       if (return_status != RTEMS_SUCCESSFUL)
110         {
111           rtems_semaphore_obtain (results_sem,
112                                   RTEMS_WAIT,
113                                   RTEMS_NO_TIMEOUT);
114           set_producer_error (task_id, system);
115           error = RTEMS_SIGNAL_SEND_ERROR;
116           show_test_results (error, return_status);
117           analyse_tasks_results ();
118           exit (-1);
119         }
120       consumers_info[consumers_index][system].signals_sent[signal]++;
121
122       /* Get next index (circular array) */
123       consumers_index = (++consumers_index) % NUMBER_OF_CONSUMERS;
124       /* Unblock waiting consumer */
125       return_status = rtems_semaphore_release (sem_id);
126       if (return_status != RTEMS_SUCCESSFUL)
127         {
128           rtems_semaphore_obtain (results_sem,
```

```
129                                     RTEMS_WAIT,
130                                     RTEMS_NO_TIMEOUT);
131             set_producer_error (task_id, system);
132             error = RTEMS_SEMAPHORE_RELEASE_ERROR;
133             show_test_results (error, return_status);
134             analyse_tasks_results ();
135             exit (-1);
136           }
137
138         rtems_task_wake_after (RTEMS_YIELD_PROCESSOR);
139         signal++;
140         if (signal == 32)
141           {
142             signal = 0;
143           }
144       }
145   }
```

sgn-consumer.c

```
  1  /*****************************************************************************
  2   SRC-MODULE : Signal Manager Workload
  3   MODULE-VERS : N/A
  4
  5             $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-sgn/sgn-consumer.c,v $
  6   $Id: sgn-consumer.c,v 1.2 2003/10/24 13:21:15 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Signal
 20   Manager. This file contains the consumer task for this workload.
 21
 22   CREATED ON : 02-10-2003
 23   CHANGED ON : $Date: 2003/10/24 13:21:15 $
 24   CHANGED BY : $Author: lhenriques $
 25
 26   $Revision: 1.2 $
 27   STICKY TAG : $Name:  $
 28
 29   INSPECTED ON:
 30   MODERATOR :
 31
 32   TABLES : none.
 33
 34   HISTORY
 35   $Log: sgn-consumer.c,v $
 36   Revision 1.2  2003/10/24 13:21:15  lhenriques
```

```
37   Added a call to the analysis function.
38
39   Revision 1.1  2003/10/10 15:42:39  lhenriques
40   First version of the stress-testing workload for the signal manager.
41
42
43   ***************************************************************************/
44
45  #include <bsp.h>
46  #include <stdio.h>
47  #include "rtems-cmp-cl-sgn.h"
48
49  /*
50   * Signal handler that will signal the semaphore
51   */
52  rtems_asr consumer_signal_handler (rtems_signal_set signals)
53  {
54    rtems_status_code return_status = RTEMS_NOT_DEFINED;
55    rtems_unsigned32 error = NO_ERROR;
56    rtems_id task_id;
57    consumer_info * this_consumer = NULL;
58    int sig = 0;
59    int index, system_index;
60
61    return_status = rtems_task_ident (RTEMS_SELF,
62                                      RTEMS_SEARCH_ALL_NODES,
63                                      &task_id);
64    if (return_status != RTEMS_SUCCESSFUL)
65      {
66        rtems_semaphore_obtain (results_sem,
67                                RTEMS_WAIT,
68                                RTEMS_NO_TIMEOUT);
69        error = RTEMS_TASK_IDENT_ERROR;
70        show_test_results (error, return_status);
71        analyse_tasks_results ();
72        exit (-1);
73      }
74
75    /* Consumer information */
76    for (system_index = 0; (system_index < NUMBER_OF_SYSTEMS) && (this_consumer ==
NULL); system_index++)
77      {
78         for (index = 0; (index < NUMBER_OF_CONSUMERS) && (this_consumer == NULL);
index++)
79          {
80            if (consumers_info[index][system_index].task_id == task_id)
81              {
82                this_consumer = &consumers_info[index][system_index];
83              }
84          }
85      }
86    if (this_consumer == NULL)
87      {
88        rtems_semaphore_obtain (results_sem,
89                                RTEMS_WAIT,
90                                RTEMS_NO_TIMEOUT);
91        error = RTEMS_GET_CONSUMER_INFO_ERROR;
92        show_test_results (error, 0);
```

```
 93        analyse_tasks_results ();
 94        exit (-1);
 95      }
 96    /* Check which signals have arrived */
 97    while (signals != 0)
 98      {
 99        if (signals & 0x00000001)
100          {
101            this_consumer->signals_received[sig]++;
102          }
103        signals = signals >> 1;
104        sig++;
105      }
106  }
107
108  void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
109  {
110    int i;
111    for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
112      {
113        if (consumers_info[i][system].task_id == task_id)
114          {
115            consumers_info[i][system].task_id = -1;
116            return;
117          }
118      }
119       printf ("[ERROR] Could not obtain the consumer %d index in consumers
array.\n", task_id);
120  }
121
122  /*
123   * This task will wait forever in the resource (a semaphore).
124   * It exits only if an error occurs in the return
125   * status of the wait directive.
126   */
127  rtems_task consumer_task (rtems_task_argument consumer_arg)
128  {
129    /* Get system number index for the semaphore array */
130    rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
131    rtems_status_code return_status = RTEMS_NOT_DEFINED;
132    rtems_unsigned32 error = NO_ERROR;
133    rtems_id task_id;
134    rtems_id sem_id = 0;
135    consumer_info * this_info = NULL;
136    int index = -1;
137    int i;
138
139    return_status = rtems_task_ident (RTEMS_SELF,
140                                      RTEMS_SEARCH_ALL_NODES,
141                                      &task_id);
142    if (return_status != RTEMS_SUCCESSFUL)
143      {
144        rtems_semaphore_obtain (results_sem,
145                                RTEMS_WAIT,
146                                RTEMS_NO_TIMEOUT);
147        error = RTEMS_TASK_IDENT_ERROR;
148        show_test_results (error, return_status);
149        analyse_tasks_results ();
```

```
150         exit (-1);
151       }
152
153     /* Get semaphore ID */
154     for (i = 0; (i < NUMBER_OF_CONSUMERS) && (index == -1); i++)
155       {
156         if (consumers_info[i][system].task_id == task_id)
157           {
158             index = i;
159             this_info = &consumers_info[i][system];
160           }
161       }
162     if (index == -1)
163       {
164         rtems_semaphore_obtain (results_sem,
165                                 RTEMS_WAIT,
166                                 RTEMS_NO_TIMEOUT);
167         error = RTEMS_GET_CONSUMER_INFO_ERROR;
168         show_test_results (error, 0);
169         analyse_tasks_results ();
170         exit (-1);
171       }
```

B.5. Interrupt Manager Workload

rtems-cmp-cl-int.h

```
1  /*****************************************************************************
2   SRC-MODULE : Interrupt Manager Workload
3   MODULE-VERS : N/A
4
5             $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-int/rtems-cmp-cl-int.h,v $
6   $Id: rtems-cmp-cl-int.h,v 1.2 2003/10/24 13:18:15 lhenriques Exp $
7   $State: Exp $
8   $Locker:  $
9
10  Copyright (c) Critical Software (www.criticalsoftware.com)
11
12  SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14  OS-TYPE : RTEMS 4.5.0
15
16  AUTHOR : lhenriques
17
18  KEYWORDS : ----
19  PURPOSE : Stress the RTEMS Classical API for the Interrupt Manager.
20
21  CREATED ON : 08-10-2003
22  CHANGED ON : $Date: 2003/10/24 13:18:15 $
23  CHANGED BY : $Author: lhenriques $
24
25  $Revision: 1.2 $
26  STICKY TAG : $Name:  $
27
28  INSPECTED ON:
```

```
 29  MODERATOR :
 30
 31  TABLES : none.
 32
 33  HISTORY
 34  $Log: rtems-cmp-cl-int.h,v $
 35  Revision 1.2  2003/10/24 13:18:15  lhenriques
 36  Changed default parameters.
 37
 38  Revision 1.1  2003/10/10 15:44:00  lhenriques
 39  First version of the stress-testing workload for the interrupt manager.
 40
 41
 42  ***************************************************************************/
 43
 44 /*
 45    * The following definitions are the parameters that shall be changed to
generated
 46  * diferent loads on the target.
 47  */
 48
 49 /* Number of producer tasks */
 50 #define NUMBER_OF_PRODUCERS 64
 51
 52 /* Stack size for the producers tasks */
 53 #define PRODUCERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
 54
 55 /* Producers priority */
 56 #define PRODUCERS_PRIORITY 1
 57
 58 /* Producers tasks mode */
 59 #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
 60
 61 /* Producers tasks attributes */
 62 #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 63
 64 /* Number of consumers tasks */
 65 #define NUMBER_OF_CONSUMERS 64
 66
 67 /* Stack size for the consumers tasks */
 68 #define CONSUMERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
 69
 70 /* Consumers tasks mode */
 71 #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
 72
 73 /* Consumers tasks attributes */
 74 #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 75
 76 /* Consumers priority */
 77 #define CONSUMERS_PRIORITY 2
 78
 79 /* Number of systems (producers/consumers systems) */
 80 #define NUMBER_OF_SYSTEMS 4
 81
 82 /* Number of ticks to wait until workload finishes */
 83 #define TEST_TIMEOUT 20000
 84
 85 /***************************************************************************/
```

```
 86  /* The following definitions are workload dependent.                         */
 87  /*****************************************************************************/
 88
 89  /* Number of illegal instructions each producer will generate */
 90  #define NUMBER_OF_INTERRUPTS 100
 91
 92  /* Time to wait before producing another item */
 93  #define DELAY_BETWEEN_ITEMS 0
 94
 95  /*
 96   * Enumeration with all the possible errors.
 97   */
 98  enum {
 99    NO_ERROR,
100    RTEMS_SEMAPHORE_CREATE_ERROR,
101    RTEMS_SEMAPHORE_RELEASE_ERROR,
102    RTEMS_SEMAPHORE_OBTAIN_ERROR,
103    RTEMS_TASK_CREATE_ERROR,
104    RTEMS_TASK_START_ERROR,
105    RTEMS_TASK_WAKE_AFTER_ERROR,
106    RTEMS_TASK_IDENT_ERROR
107  };
108
109  /* Arrays to store the producers and consumers IDs */
110  extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
111  extern rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
112
113  /* The semaphore where the consumers will wait. */
114  extern rtems_id sem_id;
115
116  /* Semaphore that shall be obtained in order to exit the workload */
117  extern rtems_id results_sem;
118
119  /*
120   * These are the producer and consumer tasks.
121   */
122  rtems_task producer_task (rtems_task_argument producer_arg);
123  rtems_task consumer_task (rtems_task_argument consumer_arg);
124
125  /*
126   * This function shall analyse the results in each task,
127   */
128  void analyse_tasks_results ();
129
130  /*
131   * This function simply checks the errors in the test
132  H */
133       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
134
135  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
136
137  /*
138   * This function switchs the return status parameter and prints a string with
139   * the corresponding error code.
140   */
141  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-int.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Interrupt Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-int/rtems-cmp-cl-int.c,v $
 6   $Id: rtems-cmp-cl-int.c,v 1.2 2003/10/24 13:17:54 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Interrupt Manager.
20
21   CREATED ON : 08-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:17:54 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-int.c,v $
35   Revision 1.2  2003/10/24 13:17:54  lhenriques
36   Corrected some problems in the tasks analysis.
37
38   Revision 1.1  2003/10/10 15:44:00  lhenriques
39   First version of the stress-testing workload for the interrupt manager.
40
41
42   *****************************************************************************/
43
44  #include <bsp.h>
45  #include <rtems/rtems/intr.h>
46  #include <stdio.h>
47  #include "rtems-cmp-cl-int.h"
48
49  rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
50  rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
51
52  rtems_id sem_id;
53
54  rtems_id results_sem;
```

```
 55
 56 rtems_unsigned32 memory_isr_counter = 0;
 57 rtems_unsigned32 instruction_isr_counter = 0;
 58
 59 int created_producers = 0;
 60 int created_consumers = 0;
 61
 62 /*
 63  * Fatal error handler. It is used to find out whether an halt has
 64  * occurred during the workload execution.
 65  */
 66 rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
 67                                      boolean is_internal,
 68                                      rtems_unsigned32 the_error)
 69 {
 70   printf ("A Fatal error has occurred!\n");
 71   printf ("Source: ");
 72   /* Find source */
 73   switch (the_source)
 74     {
 75     case INTERNAL_ERROR_CORE:
 76       printf ("INTERNAL_ERROR_CORE");
 77       break;
 78     case INTERNAL_ERROR_RTEMS_API:
 79       printf ("INTERNAL_ERROR_RTEMS_API");
 80       break;
 81     case INTERNAL_ERROR_POSIX_API:
 82       printf ("INTERNAL_ERROR_POSIX_API");
 83       break;
 84     case INTERNAL_ERROR_ITRON_API:
 85       printf ("INTERNAL_ERROR_ITRON_API");
 86     default:
 87       printf ("UNKOWN (%d)", the_source);
 88     }
 89   printf ("\n");
 90
 91   if (is_internal == TRUE)
 92     {
 93       printf ("It is an INTERNAL error.\n");
 94
 95       printf ("Error: ");
 96       /* Find the error itself */
 97       switch (the_error)
 98         {
 99         case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
100           printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
101           break;
102         case INTERNAL_ERROR_NO_CPU_TABLE:
103           printf ("INTERNAL_ERROR_NO_CPU_TABLE");
104           break;
105         case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
106           printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
107           break;
108         case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
109           printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
110           break;
111         case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
112           printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
```

```
113           break;
114         case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
115           printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
116           break;
117         case INTERNAL_ERROR_THREAD_EXITTED:
118           printf ("INTERNAL_ERROR_THREAD_EXITTED");
119           break;
120         case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
121           printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
122           break;
123         case INTERNAL_ERROR_INVALID_NODE:
124           printf ("INTERNAL_ERROR_INVALID_NODE");
125           break;
126         case INTERNAL_ERROR_NO_MPCI:
127           printf ("INTERNAL_ERROR_NO_MPCI");
128           break;
129         case INTERNAL_ERROR_BAD_PACKET:
130           printf ("INTERNAL_ERROR_BAD_PACKET");
131           break;
132         case INTERNAL_ERROR_OUT_OF_PACKETS:
133           printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
134           break;
135         case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
136           printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
137           break;
138         case INTERNAL_ERROR_OUT_OF_PROXIES:
139           printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
140           break;
141         case INTERNAL_ERROR_INVALID_GLOBAL_ID:
142           printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
143           break;
144         case INTERNAL_ERROR_BAD_STACK_HOOK:
145           printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
146           break;
147         case INTERNAL_ERROR_BAD_ATTRIBUTES:
148           printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
149           break;
150         default:
151           printf ("UNKNOWN (%d)", the_error);
152         }
153       printf ("\n");
154
155     }
156   else
157     {
158       printf ("It is NOT an internal error.\n");
159       /* Assume an RTEMS Classic API error... */
160       show_error_code (the_error);
161     }
162   analyse_tasks_results ();
163   exit (-1);
164 }
165
166 /*
167  * This structure defines the user extensions entry points
168  */
169 rtems_extensions_table user_extensions =
170   {
```

```
171     NULL,               /* task creation extension */
172     NULL,               /* task start extension */
173     NULL,               /* task restart extension */
174     NULL,               /* task delete extension */
175     NULL,               /* task switch extension */
176     NULL,               /* task begin extension */
177     NULL,               /* task exitted extension */
178     fatal_error_handler /* fatal error extension */
179   };
180
181 /*
182  * Handler to the illegal instruction interrupt
183  */
184 rtems_isr illegal_instruction_ISR (rtems_vector_number ignored)
185 {
186   rtems_status_code return_status = RTEMS_NOT_DEFINED;
187   rtems_unsigned32 error = NO_ERROR;
188   return_status = rtems_semaphore_release (sem_id);
189   if (return_status != RTEMS_SUCCESSFUL)
190     {
191       rtems_semaphore_obtain (results_sem,
192                               RTEMS_WAIT,
193                               RTEMS_NO_TIMEOUT);
194       error = RTEMS_SEMAPHORE_RELEASE_ERROR;
195       show_test_results (error, return_status);
196       analyse_tasks_results ();
197       exit (-1);
198     }
199   instruction_isr_counter++;
200 }
201
202 /*
203  * Handler to the memory address not aligned interrupt
204  */
205 rtems_isr memory_not_aligned_ISR (rtems_vector_number ignored)
206 {
207   rtems_status_code return_status = RTEMS_NOT_DEFINED;
208   rtems_unsigned32 error = NO_ERROR;
209   return_status = rtems_semaphore_release (sem_id);
210   if (return_status != RTEMS_SUCCESSFUL)
211     {
212       rtems_semaphore_obtain (results_sem,
213                               RTEMS_WAIT,
214                               RTEMS_NO_TIMEOUT);
215       error = RTEMS_SEMAPHORE_RELEASE_ERROR;
216       show_test_results (error, return_status);
217       analyse_tasks_results ();
218       exit (-1);
219     }
220   memory_isr_counter++;
221 }
222 /*
223  * Init task is the first to be executed.
224  */
225 rtems_task Init(rtems_task_argument ignored)
226 {
227   rtems_unsigned32 error = NO_ERROR;
228   rtems_status_code return_status = RTEMS_NOT_DEFINED;
```

```
229    rtems_unsigned32 system_index;
230    rtems_unsigned32 index;
231    rtems_unsigned32 max_index;
232    rtems_name results_sem_name;
233    rtems_name table_name;
234    rtems_id table_id;
235    rtems_name sem_name;
236
237    /* RTEMS determines if a trap is sychronous or asynchronous
238     * by adding 0x100 to the sychronous traps.
239     */
240         rtems_vector_number  illegal_instruction_vec_num  =  0x102;  /*  illegal
instruction */
241    rtems_vector_number memory_not_aligned_vec_num = 0x107;  /* memory address not
aligned */
242
243    rtems_isr_entry old_isr;
244
245    /* Add user extension table to handle fata error */
246    table_name = rtems_build_name ('U', 'S', 'E', 'R');
247    return_status = rtems_extension_create (table_name,
248                                            &user_extensions,
249                                            &table_id);
250
251    if (return_status != RTEMS_SUCCESSFUL)
252      {
253              printf  ("Error  registering  an  user  handler  to  the  fatal  error
extension.\n");
254        show_error_code (return_status);
255        exit (-1);
256      }
257
258    /* Create the results semaphore */
259    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
260    return_status = rtems_semaphore_create (results_sem_name,
261                                            1, /* Only one task can exit */
262                                            RTEMS_DEFAULT_ATTRIBUTES,
263                                            RTEMS_NO_PRIORITY,
264                                            &results_sem);
265    /* If this fails... exit */
266    if (return_status != RTEMS_SUCCESSFUL)
267      {
268        printf ("Error creating semaphore for exit point.\n");
269        show_error_code (return_status);
270        exit (-1);
271      }
272
273    /* Add interrupt handler to the illegal instruction trap */
274    return_status = rtems_interrupt_catch (illegal_instruction_ISR,
275                                           illegal_instruction_vec_num,
276                                           &old_isr);
277    if (return_status != RTEMS_SUCCESSFUL)
278      {
279        printf ("Error installing illegal instruction interrupt handler.\n");
280        show_error_code (return_status);
281        exit (-1);
282      }
283
```

```
284    /* Add interrupt handler to the illegal instruction trap */
285    return_status = rtems_interrupt_catch (memory_not_aligned_ISR,
286                                           memory_not_aligned_vec_num,
287                                           &old_isr);
288    if (return_status != RTEMS_SUCCESSFUL)
289      {
290        printf ("Error installing memory not aligned interrupt handler.\n");
291        show_error_code (return_status);
292        exit (-1);
293      }
294
295    sem_name = rtems_build_name ('S', 'M', 'P', ' ');
296    return_status = rtems_semaphore_create (sem_name,
297                                            0,
298                                            RTEMS_DEFAULT_ATTRIBUTES,
299                                            RTEMS_NO_PRIORITY,
300                                            &sem_id);
301
302    if (return_status != RTEMS_SUCCESSFUL)
303      {
304        printf ("Error creating semaphore.\n");
305        show_error_code (return_status);
306        exit (-1);
307      }
308
309    /* Create all the NUMBER_OF_SYSTEMS systems. */
310    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
311      {
312        rtems_name producer_task_name;
313        rtems_name consumer_task_name;
314
315        /* Create all the producers and consumers for this system */
316                  max_index  =  (NUMBER_OF_PRODUCERS  <=  NUMBER_OF_CONSUMERS  ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
317        for (index = 0; index < max_index; index++)
318          {
319            if (index < NUMBER_OF_PRODUCERS)
320              {
321                /* Create a producer */
322                producer_task_name = index + 1;
323                return_status = rtems_task_create (producer_task_name,
324                                                   PRODUCERS_PRIORITY,
325                                                   PRODUCERS_TASK_STACK_SIZE,
326                                                   PRODUCERS_TASK_MODE,
327                                                   PRODUCERS_TASK_ATTR,
328
&producers_array[index][system_index]);
329                if (return_status != RTEMS_SUCCESSFUL)
330                  {
331                    rtems_semaphore_obtain (results_sem,
332                                            RTEMS_WAIT,
333                                            RTEMS_NO_TIMEOUT);
334                    error = RTEMS_TASK_CREATE_ERROR;
335                    show_test_results (error, return_status);
336                    analyse_tasks_results ();
337                    exit (-1);
338                  }
339
```

```
   340                                             return_status   =   rtems_task_start
(producers_array[index][system_index],
   341                                                producer_task,
   342                                                system_index);
   343               if (return_status != RTEMS_SUCCESSFUL)
   344                 {
   345                   rtems_semaphore_obtain (results_sem,
   346                                           RTEMS_WAIT,
   347                                           RTEMS_NO_TIMEOUT);
   348                   error = RTEMS_TASK_START_ERROR;
   349                   show_test_results (error, return_status);
   350                   analyse_tasks_results ();
   351                   exit (-1);
   352                 }
   353               created_producers++;
   354             }
   355           if (index < NUMBER_OF_CONSUMERS)
   356             {
   357               /* Create a consumer */
   358               consumer_task_name = index + 1;
   359               return_status = rtems_task_create (consumer_task_name,
   360                                                  CONSUMERS_PRIORITY,
   361                                                  CONSUMERS_TASK_STACK_SIZE,
   362                                                  CONSUMERS_TASK_MODE,
   363                                                  CONSUMERS_TASK_ATTR,
   364
&consumers_array[index][system_index]);
   365               if (return_status != RTEMS_SUCCESSFUL)
   366                 {
   367                   rtems_semaphore_obtain (results_sem,
   368                                           RTEMS_WAIT,
   369                                           RTEMS_NO_TIMEOUT);
   370                   error = RTEMS_TASK_CREATE_ERROR;
   371                   show_test_results (error, return_status);
   372                   analyse_tasks_results ();
   373                   exit (-1);
   374                 }
   375
   376                                             return_status   =   rtems_task_start
(consumers_array[index][system_index],
   377                                                consumer_task,
   378                                                system_index);
   379               if (return_status != RTEMS_SUCCESSFUL)
   380                 {
   381                   rtems_semaphore_obtain (results_sem,
   382                                           RTEMS_WAIT,
   383                                           RTEMS_NO_TIMEOUT);
   384                   error = RTEMS_TASK_START_ERROR;
   385                   show_test_results (error, return_status);
   386                   analyse_tasks_results ();
   387                   exit (-1);
   388                 }
   389               created_consumers++;
   390             }
   391         }
   392     }
   393
   394   /* Wait for all tasks or for an error. */
```

```
395    return_status = rtems_task_wake_after (TEST_TIMEOUT);
396
397    if (return_status != RTEMS_SUCCESSFUL)
398      {
399        error = RTEMS_TASK_WAKE_AFTER_ERROR;
400      }
401
402    rtems_semaphore_obtain (results_sem,
403                            RTEMS_WAIT,
404                            RTEMS_NO_TIMEOUT);
405    show_test_results (error, return_status);
406    analyse_tasks_results ();
407
408    exit (0);
409  }
410
411  void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
412  {
413    printf ("================================================\n");
414    printf ("Test Paramenters\n");
415    printf ("================================================\n");
416    printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
417    printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
418    printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
419    printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
420    printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
421    printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
422    printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
423    printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
424    printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
425    printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
426    printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
427    printf ("================================================\n");
428    /* Workload specific parameters */
429             printf   ("Number   of   illegal   instruction   interrupts:   %u\n",
instruction_isr_counter);
430          printf  ("Number  of  memory  address  not  aligned  interrupts:  %u\n",
memory_isr_counter);
431       printf ("Total number of interruptions: %u\n", (instruction_isr_counter +
memory_isr_counter));
432         printf  ("Number  of  expected  interrupts:  %u\n",  (NUMBER_OF_SYSTEMS  *
NUMBER_OF_PRODUCERS * NUMBER_OF_INTERRUPTS * 2))
;
433
434    if (error == NO_ERROR)
435      {
436        printf ("Systems created sucessfully:\n");
437      }
438    else
439      {
440        parse_error (error, return_status);
441      }
442  }
443
444  void analyse_tasks_results ()
445  {
446    int system_index, index;
447    int number_of_consumers = 0;
```

```
   448    int number_of_producers = 0;
   449    int failed_consumers = 0;
   450    int failed_consumer_index = -1;
   451    int failed_consumer_system = -1;
   452    int failed_producers = 0;
   453    int failed_producer_index = -1;
   454    int failed_producer_system = -1;
   455
   456    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
   457      {
   458        /* Analyse producers array */
   459        for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
   460          {
   461            if (producers_array[index][system_index] != 0)
   462              {
   463                number_of_producers++;
   464                     if (rtems_get_class (producers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
   465                  {
   466                    failed_producers++;
   467                    if (failed_producer_index == -1)
   468                      {
   469                        failed_producer_index = index;
   470                        failed_producer_system = system_index;
   471                      }
   472                  }
   473              }
   474          }
   475        /* Analyse consumers array */
   476        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
   477          {
   478            if (consumers_array[index][system_index] != 0)
   479              {
   480                number_of_consumers++;
   481                     if (rtems_get_class (consumers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
   482                  {
   483                    failed_consumers++;
   484                    if (failed_consumer_index == -1)
   485                      {
   486                        failed_consumer_index = index;
   487                        failed_consumer_system = system_index;
   488                      }
   489                  }
   490              }
   491          }
   492      }
   493         printf ("Total number of successfully created producers: %d\n",
number_of_producers);
   494    printf ("Number of failed producers: %d\n", failed_producers);
   495    if (failed_producers > 0)
   496      {
   497             printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
   498              printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
   499      }
```

```
500         printf ("Total number of successfully created consumers: %d\n",
number_of_consumers);
501    printf ("Number of failed consumers: %d\n", failed_consumers);
502    if (failed_consumers > 0)
503      {
504             printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
505              printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
506      }
507  }
508
509  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
510  {
511        switch (error)
512          {
513          case RTEMS_SEMAPHORE_CREATE_ERROR:
514            printf ("Error creating semaphore.\n");
515            break;
516          case RTEMS_SEMAPHORE_RELEASE_ERROR:
517            printf ("Error releasing a semaphore.\n");
518            break;
519          case RTEMS_SEMAPHORE_OBTAIN_ERROR:
520            printf ("Error obtaining a semaphore.\n");
521            break;
522          case RTEMS_TASK_CREATE_ERROR:
523            printf ("Error creating task.\n");
524            break;
525          case RTEMS_TASK_START_ERROR:
526            printf ("Error starting task.\n");
527            break;
528          case RTEMS_TASK_WAKE_AFTER_ERROR:
529            printf ("Error waiting for tasks to finish.\n");
530            break;
531          case RTEMS_TASK_IDENT_ERROR:
532            printf ("Error obtaining thread ID.\n");
533            break;
534          default:
535            printf ("Unknown failure. Possible bug in workload.\n");
536            break;
537          }
538        show_error_code (return_status);
539  }
540
541  void show_error_code (rtems_status_code return_status)
542  {
543    printf ("Error code: ");
544    switch (return_status)
545      {
546      case RTEMS_TASK_EXITTED:
547        printf ("RTEMS_TASK_EXITTED");
548        break;
549      case RTEMS_MP_NOT_CONFIGURED:
550        printf ("RTEMS_MP_NOT_CONFIGURED");
551        break;
552      case RTEMS_INVALID_NAME:
553        printf ("RTEMS_INVALID_NAME");
554        break;
```

```
555       case RTEMS_INVALID_ID:
556         printf ("RTEMS_INVALID_ID");
557         break;
558       case RTEMS_TOO_MANY:
559         printf ("RTEMS_TOO_MANY");
560         break;
561       case RTEMS_TIMEOUT:
562         printf ("RTEMS_TIMEOUT");
563         break;
564       case RTEMS_OBJECT_WAS_DELETED:
565         printf ("RTEMS_OBJECT_WAS_DELETED");
566         break;
567       case RTEMS_INVALID_SIZE:
568         printf ("RTEMS_INVALID_SIZE");
569         break;
570       case RTEMS_INVALID_ADDRESS:
571         printf ("RTEMS_INVALID_ADDRESS");
572         break;
573       case RTEMS_INVALID_NUMBER:
574         printf ("RTEMS_INVALID_NUMBER");
575         break;
576       case RTEMS_NOT_DEFINED:
577         printf ("RTEMS_NOT_DEFINED");
578         break;
579       case RTEMS_RESOURCE_IN_USE:
580         printf ("RTEMS_RESOURCE_IN_USE");
581         break;
582       case RTEMS_UNSATISFIED:
583         printf ("RTEMS_UNSATISFIED");
584         break;
585       case RTEMS_INCORRECT_STATE:
586         printf ("RTEMS_INCORRECT_STATE");
587         break;
588       case RTEMS_ALREADY_SUSPENDED:
589         printf ("RTEMS_ALREADY_SUSPENDED");
590         break;
591       case RTEMS_ILLEGAL_ON_SELF:
592         printf ("RTEMS_ILLEGAL_ON_SELF");
593         break;
594       case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
595         printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
596         break;
597       case RTEMS_CALLED_FROM_ISR:
598         printf ("RTEMS_CALLED_FROM_ISR");
599         break;
600       case RTEMS_INVALID_PRIORITY:
601         printf ("RTEMS_INVALID_PRIORITY");
602         break;
603       case RTEMS_INVALID_CLOCK:
604         printf ("RTEMS_INVALID_CLOCK");
605         break;
606       case RTEMS_INVALID_NODE:
607         printf ("RTEMS_INVALID_NODE");
608         break;
609       case RTEMS_NOT_CONFIGURED:
610         printf ("RTEMS_NOT_CONFIGURED");
611         break;
612       case RTEMS_NOT_OWNER_OF_RESOURCE:
```

```
613         printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
614         break;
615       case RTEMS_NOT_IMPLEMENTED:
616         printf ("RTEMS_NOT_IMPLEMENTED");
617         break;
618       case RTEMS_INTERNAL_ERROR:
619         printf ("RTEMS_INTERNAL_ERROR");
620         break;
621       case RTEMS_NO_MEMORY:
622         printf ("RTEMS_NO_MEMORY");
623         break;
624       case RTEMS_IO_ERROR:
625         printf ("RTEMS_IO_ERROR");
626         break;
627       case RTEMS_PROXY_BLOCKING:
628         printf ("RTEMS_PROXY_BLOCKING");
629         break;
630       default:
631         printf ("UNKNOWN ERROR CODE");
632       }
633     printf ("\n");
634   }
635
636   /* configuration information */
637
638   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
639   #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
640
641   #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
642   #define CONFIGURE_INIT_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 2)
643
644    #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
645
646   #define CONFIGURE_MAXIMUM_SEMAPHORES 2
647
648   #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
649
650   #define CONFIGURE_INIT
651
652   #include <confdefs.h>
653
654   /* end of file */
```

int-producer.c

```
1   /*************************************************************************
2    SRC-MODULE : Interrupt Manager Workload
3    MODULE-VERS : N/A
4
5             $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-int/int-producer.c,v $
6    $Id: int-producer.c,v 1.2 2003/10/24 13:17:40 lhenriques Exp $
7    $State: Exp $
8    $Locker:  $
9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
```

```
11
12  SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14  OS-TYPE : RTEMS 4.5.0
15
16  AUTHOR : lhenriques
17
18  KEYWORDS : ----
19  PURPOSE : Stress the RTEMS Classical API for the Interrupt
20  Manager. This file contains the producer task for this workload.
21
22  CREATED ON : 08-10-2003
23  CHANGED ON : $Date: 2003/10/24 13:17:40 $
24  CHANGED BY : $Author: lhenriques $
25
26  $Revision: 1.2 $
27  STICKY TAG : $Name:  $
28
29  INSPECTED ON:
30  MODERATOR :
31
32  TABLES : none.
33
34  HISTORY
35  $Log: int-producer.c,v $
36  Revision 1.2  2003/10/24 13:17:40  lhenriques
37  Removed unused code.
38
39  Revision 1.1  2003/10/10 15:44:00  lhenriques
40  First version of the stress-testing workload for the interrupt manager.
41
42
43  ***************************************************************************/
44
45 #include <bsp.h>
46 #include <stdio.h>
47 #include "rtems-cmp-cl-int.h"
48
49
50 void set_producer_error (rtems_id task_id, rtems_unsigned32 system)
51 {
52   int i;
53   for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
54     {
55       if (producers_array[i][system] == task_id)
56         {
57           producers_array[i][system] = -1;
58           return;
59         }
60     }
61      printf ("[ERROR] Could not obtain the producer %d index in producers
array.\n", task_id);
62 }
63
64 rtems_task producer_task (rtems_task_argument producer_arg)
65 {
66   /* Number of rtems_message_queue_send calls */
67   rtems_signed32 count = NUMBER_OF_INTERRUPTS;
```

```
  rtems_status_code return_status = RTEMS_NOT_DEFINED;
  rtems_unsigned32 error = NO_ERROR;
  rtems_id task_id;

  return_status = rtems_task_ident (RTEMS_SELF,
                                    RTEMS_SEARCH_ALL_NODES,
                                    &task_id);
  if (return_status != RTEMS_SUCCESSFUL)
    {
      rtems_semaphore_obtain (results_sem,
                              RTEMS_WAIT,
                              RTEMS_NO_TIMEOUT);
      error = RTEMS_TASK_IDENT_ERROR;
      show_test_results (error, return_status);
      analyse_tasks_results ();
      exit (-1);
    }

  while (count-- > 0)
    {
      /* Cause a memory not aligned interrupt */
      asm ("sethi %hi(0x01f80000), %g7");
      asm ("or %g7, 0x0064, %g7");
      asm ("ldd [%g7], %g7");

      /* Cause an illegal instruction interrupt */
      asm ("unimp");

      /* Wait before sending other message */
#if 0
      return_status = rtems_task_wake_after (RTEMS_YIELD_PROCESSOR);
      if (return_status != RTEMS_SUCCESSFUL)
        {
          rtems_semaphore_obtain (results_sem,
                                  RTEMS_WAIT,
                                  RTEMS_NO_TIMEOUT);
          set_producer_error (task_id, system);
          error = RTEMS_TASK_WAKE_AFTER_ERROR;
          show_test_results (error, return_status);
          analyse_tasks_results ();
          exit (-1);
          }
#endif 0
    }
}
```

int-consumer.c

```
/*****************************************************************************
 SRC-MODULE : Interrupt Manager Workload
 MODULE-VERS : N/A

          $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-int/int-consumer.c,v $
 $Id: int-consumer.c,v 1.1 2003/10/10 15:44:00 lhenriques Exp $
 $State: Exp $
 $Locker:  $
```

```
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Interrupt
20   Manager. This file contains the consumer task for this workload.
21
22   CREATED ON : 08-10-2003
23   CHANGED ON : $Date: 2003/10/10 15:44:00 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: int-consumer.c,v $
36   Revision 1.1  2003/10/10 15:44:00  lhenriques
37   First version of the stress-testing workload for the interrupt manager.
38
39
40   ******************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-int.h"
45
46  void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
47  {
48    int i;
49    for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
50      {
51        if (consumers_array[i][system] == task_id)
52          {
53            consumers_array[i][system] = -1;
54            return;
55          }
56      }
57       printf ("[ERROR] Could not obtain the consumer %d index in consumers
array.\n", task_id);
58  }
59
60  rtems_task consumer_task (rtems_task_argument consumer_arg)
61  {
62    /* Get system number index for the message queues array */
63    rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
64    rtems_status_code return_status = RTEMS_NOT_DEFINED;
65    rtems_unsigned32 error = NO_ERROR;
```

```
 66    rtems_id task_id;
 67
 68    return_status = rtems_task_ident (RTEMS_SELF,
 69                                      RTEMS_SEARCH_ALL_NODES,
 70                                      &task_id);
 71    if (return_status != RTEMS_SUCCESSFUL)
 72      {
 73        rtems_semaphore_obtain (results_sem,
 74                                RTEMS_WAIT,
 75                                RTEMS_NO_TIMEOUT);
 76        error = RTEMS_TASK_IDENT_ERROR;
 77        show_test_results (error, return_status);
 78        analyse_tasks_results ();
 79        exit (-1);
 80      }
 81    while (TRUE)
 82      {
 83        return_status = rtems_semaphore_obtain (sem_id,
 84                                                RTEMS_WAIT,
 85                                                RTEMS_NO_TIMEOUT);
 86        if (return_status != RTEMS_SUCCESSFUL)
 87          {
 88            rtems_semaphore_obtain (results_sem,
 89                                    RTEMS_WAIT,
 90                                    RTEMS_NO_TIMEOUT);
 91            error = RTEMS_SEMAPHORE_OBTAIN_ERROR;
 92            set_consumer_error (task_id, system);
 93            show_test_results (error, return_status);
 94            analyse_tasks_results ();
 95            exit (-1);
 96          }
 97        /*      printf (".");
 98                fflush (stdout);
 99        */
100      }
101  }
```

B.6. Event Manager Workload

rtems-cmp-cl-evt.h

```
  1  /**************************************************************************
  2   SRC-MODULE : Event Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-evt/rtems-cmp-cl-evt.h,v $
  6   $Id: rtems-cmp-cl-evt.h,v 1.2 2003/10/24 13:16:03 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
```

```
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Event Manager.
20
21   CREATED ON : 02-10-2003
22   CHANGED ON : $Date: 2003/10/24 13:16:03 $
23   CHANGED BY : $Author: lhenriques $
24
25   $Revision: 1.2 $
26   STICKY TAG : $Name:  $
27
28   INSPECTED ON:
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-evt.h,v $
35   Revision 1.2  2003/10/24 13:16:03  lhenriques
36   Changed default parameters and corrected arrays.
37
38   Revision 1.1  2003/10/10 15:44:26  lhenriques
39   First version of the stress-testing workload for the event manager.
40
41
42   ***************************************************************************/
43
44  /*
45     * The following definitions are the parameters that shall be changed to
generated
46   * diferent loads on the target.
47   */
48
49  /* Number of producer tasks */
50  #define NUMBER_OF_PRODUCERS 64
51
52  /* Stack size for the producers tasks */
53  #define PRODUCERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
54
55  /* Producers priority */
56  #define PRODUCERS_PRIORITY 3
57
58  /* Producers tasks mode */
59  #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
60
61  /* Producers tasks attributes */
62  #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
63
64  /* Number of consumers tasks */
65  #define NUMBER_OF_CONSUMERS 64
66
67  /* Stack size for the consumers tasks */
68  #define CONSUMERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
69
70  /* Consumers tasks mode */
71  #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
```

```
 72
 73  /* Consumers tasks attributes */
 74  #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 75
 76  /*
 77   * Consumers priority
 78   * NOTE: for this workload, it shall always be higher than the producer's
 79   */
 80  #define CONSUMERS_PRIORITY 2
 81
 82  /* Number of systems (producers/consumers systems) */
 83  #define NUMBER_OF_SYSTEMS 4
 84
 85  /* Number of ticks to wait until workload finishes */
 86  #define TEST_TIMEOUT 20000
 87
 88  /*************************************************************************/
 89  /* The following definitions are workload dependent.                     */
 90  /*************************************************************************/
 91
 92  /* Number of signals each producer has to send */
 93  #define NUMBER_OF_EVENTS 320
 94
 95  /* Time to wait before producing another item */
 96  #define DELAY_BETWEEN_ITEMS 1
 97
 98  /*
 99   * Enumeration with all the possible errors.
100   */
101  enum {
102    NO_ERROR,
103    RTEMS_SEMAPHORE_CREATE_ERROR,
104    RTEMS_SEMAPHORE_OBTAIN_ERROR,
105    RTEMS_SEMAPHORE_RELEASE_ERROR,
106    RTEMS_SEMAPHORE_FLUSH_ERROR,
107    RTEMS_TASK_CREATE_ERROR,
108    RTEMS_TASK_START_ERROR,
109    RTEMS_TASK_WAKE_AFTER_ERROR,
110    RTEMS_TASK_IDENT_ERROR,
111    RTEMS_EVENT_SEND_ERROR,
112    RTEMS_EVENT_RECEIVE_ERROR,
113    RTEMS_GET_CONSUMER_INFO_ERROR
114  };
115
116  typedef struct {
117    rtems_id task_id;
118    rtems_unsigned32 events_sent[32];
119    rtems_unsigned32 events_received[32];
120  } consumer_info;
121
122  /* Arrays to store the producers and consumers IDs */
123  extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
124
125  /* Array with consumers semaphores. */
126  extern consumer_info consumers_info[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
127
128  /* Semaphore that shall be obtained in order to exit the workload */
129  extern rtems_id results_sem;
```

```
130
131  /*
132   * These are the producer and consumer tasks.
133   */
134  rtems_task producer_task (rtems_task_argument producer_arg);
135  rtems_task consumer_task (rtems_task_argument consumer_arg);
136
137  /*
138   * This function shall analyse the results in each task,
139   */
140  void analyse_tasks_results ();
141
142  /*
143   * This function simply checks the errors in the test
144  H */
145       void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
146
147  void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
148
149  /*
150   * This function switchs the return status parameter and prints a string with
151   * the corresponding error code.
152   */
153  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-evt.c

```
1  /*****************************************************************************
2   SRC-MODULE : Event Manager Workload
3   MODULE-VERS : N/A
4
5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-evt/rtems-cmp-cl-evt.c,v $
6   $Id: rtems-cmp-cl-evt.c,v 1.2 2003/10/24 13:16:23 lhenriques Exp $
7   $State: Exp $
8   $Locker:  $
9
10  Copyright (c) Critical Software (www.criticalsoftware.com)
11
12  SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14  OS-TYPE : RTEMS 4.5.0
15
16  AUTHOR : lhenriques
17
18  KEYWORDS : ----
19  PURPOSE : Stress the RTEMS Classical API for the Event Manager.
20
21  CREATED ON : 09-10-2003
22  CHANGED ON : $Date: 2003/10/24 13:16:23 $
23  CHANGED BY : $Author: lhenriques $
24
25  $Revision: 1.2 $
26  STICKY TAG : $Name:  $
27
28  INSPECTED ON:
```

```
29   MODERATOR :
30
31   TABLES : none.
32
33   HISTORY
34   $Log: rtems-cmp-cl-evt.c,v $
35   Revision 1.2  2003/10/24 13:16:23  lhenriques
36   Corrected some problems in the tasks analysis.
37
38   Revision 1.1  2003/10/10 15:44:26  lhenriques
39   First version of the stress-testing workload for the event manager.
40
41
42   ***************************************************************************/
43
44 #include <bsp.h>
45 #include <stdio.h>
46 #include "rtems-cmp-cl-evt.h"
47
48 rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
49
50 consumer_info consumers_info[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
51
52 rtems_id results_sem;
53
54 int created_producers = 0;
55 int created_consumers = 0;
56
57 /*
58  * Fatal error handler. It is used to find out whether an halt has
59  * occurred during the workload execution.
60  */
61 rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
62                                      boolean is_internal,
63                                      rtems_unsigned32 the_error)
64 {
65   printf ("A Fatal error has occurred!\n");
66   printf ("Source: ");
67   /* Find source */
68   switch (the_source)
69     {
70     case INTERNAL_ERROR_CORE:
71       printf ("INTERNAL_ERROR_CORE");
72       break;
73     case INTERNAL_ERROR_RTEMS_API:
74       printf ("INTERNAL_ERROR_RTEMS_API");
75       break;
76     case INTERNAL_ERROR_POSIX_API:
77       printf ("INTERNAL_ERROR_POSIX_API");
78       break;
79     case INTERNAL_ERROR_ITRON_API:
80       printf ("INTERNAL_ERROR_ITRON_API");
81     default:
82       printf ("UNKOWN (%d)", the_source);
83     }
84   printf ("\n");
85
86   if (is_internal == TRUE)
```

```
 87      {
 88        printf ("It is an INTERNAL error.\n");
 89
 90        printf ("Error: ");
 91        /* Find the error itself */
 92        switch (the_error)
 93          {
 94          case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
 95            printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
 96            break;
 97          case INTERNAL_ERROR_NO_CPU_TABLE:
 98            printf ("INTERNAL_ERROR_NO_CPU_TABLE");
 99            break;
100          case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
101            printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
102            break;
103          case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
104            printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
105            break;
106          case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
107            printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
108            break;
109          case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
110            printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
111            break;
112          case INTERNAL_ERROR_THREAD_EXITTED:
113            printf ("INTERNAL_ERROR_THREAD_EXITTED");
114            break;
115          case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
116            printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
117            break;
118          case INTERNAL_ERROR_INVALID_NODE:
119            printf ("INTERNAL_ERROR_INVALID_NODE");
120            break;
121          case INTERNAL_ERROR_NO_MPCI:
122            printf ("INTERNAL_ERROR_NO_MPCI");
123            break;
124          case INTERNAL_ERROR_BAD_PACKET:
125            printf ("INTERNAL_ERROR_BAD_PACKET");
126            break;
127          case INTERNAL_ERROR_OUT_OF_PACKETS:
128            printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
129            break;
130          case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
131            printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
132            break;
133          case INTERNAL_ERROR_OUT_OF_PROXIES:
134            printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
135            break;
136          case INTERNAL_ERROR_INVALID_GLOBAL_ID:
137            printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
138            break;
139          case INTERNAL_ERROR_BAD_STACK_HOOK:
140            printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
141            break;
142          case INTERNAL_ERROR_BAD_ATTRIBUTES:
143            printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
144            break;
```

```
145         default:
146           printf ("UNKNOWN (%d)", the_error);
147         }
148       printf ("\n");
149
150     }
151   else
152     {
153       printf ("It is NOT an internal error.\n");
154       /* Assume an RTEMS Classic API error... */
155       show_error_code (the_error);
156     }
157   analyse_tasks_results ();
158   exit (-1);
159 }
160
161 /*
162  * This structure defines the user extensions entry points
163  */
164 rtems_extensions_table user_extensions =
165   {
166     NULL,               /* task creation extension */
167     NULL,               /* task start extension */
168     NULL,               /* task restart extension */
169     NULL,               /* task delete extension */
170     NULL,               /* task switch extension */
171     NULL,               /* task begin extension */
172     NULL,               /* task exitted extension */
173     fatal_error_handler /* fatal error extension */
174   };
175
176 /*
177  * Init task is the first to be executed.
178  */
179 rtems_task Init(rtems_task_argument ignored)
180 {
181   rtems_unsigned32 error = NO_ERROR;
182   rtems_status_code return_status = RTEMS_NOT_DEFINED;
183   rtems_unsigned32 system_index;
184   rtems_unsigned32 index;
185   rtems_unsigned32 max_index;
186   rtems_name results_sem_name;
187   rtems_name table_name;
188   rtems_id table_id;
189
190   /* Add user extension table to handle fata error */
191   table_name = rtems_build_name ('U', 'S', 'E', 'R');
192   return_status = rtems_extension_create (table_name,
193                                           &user_extensions,
194                                           &table_id);
195
196   if (return_status != RTEMS_SUCCESSFUL)
197     {
198             printf ("Error registering an user handler to the fatal error
extension.\n");
199       show_error_code (return_status);
200       exit (-1);
201     }
```

```
202
203    /* Create the results semaphore */
204    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
205    return_status = rtems_semaphore_create (results_sem_name,
206                                            1, /* Only one task can exit */
207                                            RTEMS_DEFAULT_ATTRIBUTES,
208                                            RTEMS_NO_PRIORITY,
209                                            &results_sem);
210    /* If this fails... exit */
211    if (return_status != RTEMS_SUCCESSFUL)
212      {
213        printf ("Error creating semaphore for exit point.\n");
214        show_error_code (return_status);
215        exit (-1);
216      }
217
218    /* Create all the NUMBER_OF_SYSTEMS systems. */
219    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
220      {
221        rtems_name producer_task_name;
222        rtems_name consumer_task_name;
223        /* Create all the producers and consumers for this system */
224                    max_index  =  (NUMBER_OF_PRODUCERS  <=  NUMBER_OF_CONSUMERS  ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
225        for (index = 0; index < max_index; index++)
226          {
227            if (index < NUMBER_OF_PRODUCERS)
228              {
229                /* Create a producer */
230                producer_task_name = index + 1;
231                return_status = rtems_task_create (producer_task_name,
232                                                   PRODUCERS_PRIORITY,
233                                                   PRODUCERS_TASK_STACK_SIZE,
234                                                   PRODUCERS_TASK_MODE,
235                                                   PRODUCERS_TASK_ATTR,
236
&producers_array[index][system_index]);
237                if (return_status != RTEMS_SUCCESSFUL)
238                  {
239                    rtems_semaphore_obtain (results_sem,
240                                            RTEMS_WAIT,
241                                            RTEMS_NO_TIMEOUT);
242                    error = RTEMS_TASK_CREATE_ERROR;
243                    show_test_results (error, return_status);
244                    analyse_tasks_results ();
245                    exit (-1);
246                  }
247              }
248            if (index < NUMBER_OF_CONSUMERS)
249              {
250                /* Create a consumer */
251                consumer_task_name = index + 1;
252                return_status = rtems_task_create (consumer_task_name,
253                                                   CONSUMERS_PRIORITY,
254                                                   CONSUMERS_TASK_STACK_SIZE,
255                                                   CONSUMERS_TASK_MODE,
256                                                   CONSUMERS_TASK_ATTR,
```

```
   257
&((consumer_info)(consumers_info[index][system_index])).task_id);
   258                if (return_status != RTEMS_SUCCESSFUL)
   259                  {
   260                    rtems_semaphore_obtain (results_sem,
   261                                            RTEMS_WAIT,
   262                                            RTEMS_NO_TIMEOUT);
   263                    error = RTEMS_TASK_CREATE_ERROR;
   264                    show_test_results (error, return_status);
   265                    analyse_tasks_results ();
   266                    exit (-1);
   267                  }
   268              }
   269          }
   270        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
   271          {
   272                                                return_status    =    rtems_task_start
(consumers_info[index][system_index].task_id,
   273                                                 consumer_task,
   274                                                 system_index);
   275            if (return_status != RTEMS_SUCCESSFUL)
   276              {
   277                rtems_semaphore_obtain (results_sem,
   278                                        RTEMS_WAIT,
   279                                        RTEMS_NO_TIMEOUT);
   280                error = RTEMS_TASK_START_ERROR;
   281                show_test_results (error, return_status);
   282                analyse_tasks_results ();
   283                exit (-1);
   284              }
   285            created_consumers++;
   286          }
   287        for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
   288          {
   289                                                return_status    =    rtems_task_start
(producers_array[index][system_index],
   290                                                 producer_task,
   291                                                 system_index);
   292            if (return_status != RTEMS_SUCCESSFUL)
   293              {
   294                rtems_semaphore_obtain (results_sem,
   295                                        RTEMS_WAIT,
   296                                        RTEMS_NO_TIMEOUT);
   297                error = RTEMS_TASK_START_ERROR;
   298                show_test_results (error, return_status);
   299                analyse_tasks_results ();
   300                exit (-1);
   301              }
   302            created_producers++;
   303          }
   304      }
   305
   306    /* Wait for all tasks or for an error. */
   307    return_status = rtems_task_wake_after (TEST_TIMEOUT);
   308
   309    if (return_status != RTEMS_SUCCESSFUL)
   310      {
   311        error = RTEMS_TASK_WAKE_AFTER_ERROR;
```

```
312       }
313
314     rtems_semaphore_obtain (results_sem,
315                             RTEMS_WAIT,
316                             RTEMS_NO_TIMEOUT);
317     show_test_results (error, return_status);
318     analyse_tasks_results ();
319
320     exit (0);
321   }
322
323   void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
324   {
325     printf ("================================================\n");
326     printf ("Test Paramenters\n");
327     printf ("================================================\n");
328     printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
329     printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
330     printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
331     printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
332     printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
333     printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
334     printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
335     printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
336     printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
337     printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
338     printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
339     printf ("================================================\n");
340     /* Workload specific parameters */
341
342     if (error == NO_ERROR)
343       {
344         printf ("Systems created sucessfully:\n");
345       }
346     else
347       {
348         parse_error (error, return_status);
349       }
350   }
351
352   void analyse_tasks_results ()
353   {
354     int system_index, index, i;
355     rtems_unsigned32 running_consumers = 0;
356     rtems_unsigned32 running_producers = 0;
357     rtems_unsigned32 failed_consumers = 0;
358     rtems_unsigned32 failed_producers = 0;
359     int failed_consumer_index = -1;
360     int failed_consumer_system = -1;
361     int failed_producer_index = -1;
362     int failed_producer_system = -1;
363
364     for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
365       {
366         /* Analyse producers array */
367           for (index = 0; (index < NUMBER_OF_PRODUCERS) && (((system_index + 1) *
(index + 1) + running_producers) < creat
ed_producers); index++)
```

```
   368          {
   369            if (producers_array[index][system_index] != 0)
   370              {
   371                running_producers++;
   372                     if (rtems_get_class (producers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
   373                  {
   374                    failed_producers++;
   375                    if (failed_producer_index == -1)
   376                      {
   377                        failed_producer_index = index;
   378                        failed_producer_system = system_index;
   379                      }
   380                  }
   381              }
   382          }
   383        /* Analyse consumers array */
   384        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
   385          {
   386            if (consumers_info[index][system_index].task_id != 0)
   387              {
   388                running_consumers++;
   389                 if (rtems_get_class (consumers_info[index][system_index].task_id)
!= OBJECTS_RTEMS_TASKS)
   390                  {
   391                    failed_consumers++;
   392                    if (failed_consumer_index == -1)
   393                      {
   394                        failed_consumer_index = index;
   395                        failed_consumer_system = system_index;
   396                      }
   397                  }
   398              }
   399          }
   400      }
   401          printf ("Total number of successfully created producers: %d\n",
created_producers);
   402    printf ("Running producers: %d\n", running_producers);
   403     printf ("Producers that terminated successfuly: %d\n", (created_producers -
running_producers));
   404    printf ("Number of failed producers: %d\n", failed_producers);
   405    if (failed_producers > 0)
   406      {
   407             printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
   408              printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
   409      }
   410          printf ("Total number of successfully created consumers: %d\n",
created_consumers);
   411    printf ("Number of failed consumers: %d\n", failed_consumers);
   412    if (failed_consumers > 0)
   413      {
   414             printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
   415              printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
   416      }
```

```
   417    /* Show number of events received by each consumer */
   418    printf ("Analysing events sent/received...\n");
   419    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
   420      {
   421        rtems_boolean system_ok = TRUE;
   422        printf ("System %d.............", system_index);
   423        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
   424          {
   425            for (i = 0; i < 32; i++)
   426              {
   427                if (consumers_info[index][system_index].events_sent[i] !=
   428                    consumers_info[index][system_index].events_received[i])
   429                  {
   430                      printf ("\nERROR: %d RTEMS_EVENT_%d events sent to consumer
%d, only %d received.\n",
   431                              consumers_info[index][system_index].events_sent[i],
   432                              i,
   433                              index,
   434
consumers_info[index][system_index].events_received[i]);
   435                    system_ok = FALSE;
   436                  }
   437              }
   438          }
   439        if (system_ok == TRUE)
   440          {
   441            printf ("OK\n");
   442          }
   443      }
   444  }
   445
   446  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
   447  {
   448        switch (error)
   449          {
   450          case RTEMS_SEMAPHORE_CREATE_ERROR:
   451            printf ("Error creating semaphore.\n");
   452            break;
   453          case RTEMS_SEMAPHORE_OBTAIN_ERROR:
   454            printf ("Error obtaining a semaphore.\n");
   455            break;
   456          case RTEMS_SEMAPHORE_RELEASE_ERROR:
   457            printf ("Error signaling a semaphore.\n");
   458            break;
   459          case RTEMS_SEMAPHORE_FLUSH_ERROR:
   460            printf ("Error flushing semaphore.\n");
   461            break;
   462          case RTEMS_TASK_CREATE_ERROR:
   463            printf ("Error creating task.\n");
   464            break;
   465          case RTEMS_TASK_START_ERROR:
   466            printf ("Error starting task.\n");
   467            break;
   468          case RTEMS_TASK_WAKE_AFTER_ERROR:
   469            printf ("Error waiting for tasks to finish.\n");
   470            break;
   471          case RTEMS_TASK_IDENT_ERROR:
   472            printf ("Error obtaining thread ID.\n");
```

```
473             break;
474           case RTEMS_EVENT_RECEIVE_ERROR:
475             printf ("Error receiving event.\n");
476             break;
477           case RTEMS_EVENT_SEND_ERROR:
478             printf ("Error sending event.\n");
479             break;
480           case RTEMS_GET_CONSUMER_INFO_ERROR:
481             printf ("Error obtaining the consumer information.\n");
482             break;
483           default:
484             printf ("Unknown failure. Possible bug in workload.\n");
485             break;
486           }
487         show_error_code (return_status);
488 }
489
490 void show_error_code (rtems_status_code return_status)
491 {
492   printf ("Error code: ");
493   switch (return_status)
494     {
495     case RTEMS_TASK_EXITTED:
496       printf ("RTEMS_TASK_EXITTED");
497       break;
498     case RTEMS_MP_NOT_CONFIGURED:
499       printf ("RTEMS_MP_NOT_CONFIGURED");
500       break;
501     case RTEMS_INVALID_NAME:
502       printf ("RTEMS_INVALID_NAME");
503       break;
504     case RTEMS_INVALID_ID:
505       printf ("RTEMS_INVALID_ID");
506       break;
507     case RTEMS_TOO_MANY:
508       printf ("RTEMS_TOO_MANY");
509       break;
510     case RTEMS_TIMEOUT:
511       printf ("RTEMS_TIMEOUT");
512       break;
513     case RTEMS_OBJECT_WAS_DELETED:
514       printf ("RTEMS_OBJECT_WAS_DELETED");
515       break;
516     case RTEMS_INVALID_SIZE:
517       printf ("RTEMS_INVALID_SIZE");
518       break;
519     case RTEMS_INVALID_ADDRESS:
520       printf ("RTEMS_INVALID_ADDRESS");
521       break;
522     case RTEMS_INVALID_NUMBER:
523       printf ("RTEMS_INVALID_NUMBER");
524       break;
525     case RTEMS_NOT_DEFINED:
526       printf ("RTEMS_NOT_DEFINED");
527       break;
528     case RTEMS_RESOURCE_IN_USE:
529       printf ("RTEMS_RESOURCE_IN_USE");
530       break;
```

```
531       case RTEMS_UNSATISFIED:
532         printf ("RTEMS_UNSATISFIED");
533         break;
534       case RTEMS_INCORRECT_STATE:
535         printf ("RTEMS_INCORRECT_STATE");
536         break;
537       case RTEMS_ALREADY_SUSPENDED:
538         printf ("RTEMS_ALREADY_SUSPENDED");
539         break;
540       case RTEMS_ILLEGAL_ON_SELF:
541         printf ("RTEMS_ILLEGAL_ON_SELF");
542         break;
543       case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
544         printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
545         break;
546       case RTEMS_CALLED_FROM_ISR:
547         printf ("RTEMS_CALLED_FROM_ISR");
548         break;
549       case RTEMS_INVALID_PRIORITY:
550         printf ("RTEMS_INVALID_PRIORITY");
551         break;
552       case RTEMS_INVALID_CLOCK:
553         printf ("RTEMS_INVALID_CLOCK");
554         break;
555       case RTEMS_INVALID_NODE:
556         printf ("RTEMS_INVALID_NODE");
557         break;
558       case RTEMS_NOT_CONFIGURED:
559         printf ("RTEMS_NOT_CONFIGURED");
560         break;
561       case RTEMS_NOT_OWNER_OF_RESOURCE:
562         printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
563         break;
564       case RTEMS_NOT_IMPLEMENTED:
565         printf ("RTEMS_NOT_IMPLEMENTED");
566         break;
567       case RTEMS_INTERNAL_ERROR:
568         printf ("RTEMS_INTERNAL_ERROR");
569         break;
570       case RTEMS_NO_MEMORY:
571         printf ("RTEMS_NO_MEMORY");
572         break;
573       case RTEMS_IO_ERROR:
574         printf ("RTEMS_IO_ERROR");
575         break;
576       case RTEMS_PROXY_BLOCKING:
577         printf ("RTEMS_PROXY_BLOCKING");
578         break;
579       default:
580         printf ("UNKNOWN ERROR CODE");
581       }
582     printf ("\n");
583   }
584
585   /* configuration information */
586
587   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
588   #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
```

```
589
590  #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
591  #define CONFIGURE_INIT_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
592  #define CONFIGURE_EXTRA_TASK_STACKS (1 * RTEMS_MINIMUM_STACK_SIZE)
593
594   #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
595
596  #define CONFIGURE_MAXIMUM_SEMAPHORES 2
597
598  #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
599
600  #define CONFIGURE_INIT
601
602  #include <confdefs.h>
603
604  /* end of file */
```

evt-producer.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Event Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-evt/evt-producer.c,v $
 6   $Id: evt-producer.c,v 1.2 2003/10/24 13:14:59 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Event
20   Manager. This file contains the producer task for this workload.
21
22   CREATED ON : 09-10-2003
23   CHANGED ON : $Date: 2003/10/24 13:14:59 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.2 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: evt-producer.c,v $
36   Revision 1.2  2003/10/24 13:14:59  lhenriques
```

```
37   Corrected a bug in the producer.
38
39   Revision 1.1  2003/10/10 15:44:26  lhenriques
40   First version of the stress-testing workload for the event manager.
41
42
43   ******************************************************************************/
44
45  #include <bsp.h>
46  #include <stdio.h>
47  #include "rtems-cmp-cl-evt.h"
48
49  #define GET_EVENT(evt, event)                                          \
50          (evt < 32) ?                                                   \
51          (event = 1 << evt++) :                                         \
52          (event = (evt = 0) + 1)
53
54  void set_producer_result (int res, rtems_id task_id, rtems_unsigned32 system)
55  {
56    int i;
57    for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
58      {
59        if (producers_array[i][system] == task_id)
60          {
61            producers_array[i][system] = res;
62            return;
63          }
64      }
65       printf ("[ERROR] Could not obtain the producer %d index in producers
array.\n", task_id);
66  }
67
68  rtems_task producer_task (rtems_task_argument producer_arg)
69  {
70    /* Get system number index */
71    rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
72    rtems_status_code return_status = RTEMS_NOT_DEFINED;
73    /* Number of signals to send */
74    rtems_signed32 count = NUMBER_OF_EVENTS;
75    rtems_unsigned32 error = NO_ERROR;
76    rtems_id task_id;
77    rtems_id event_task_id;
78    int consumers_index = 0;
79    int event = 0;
80    rtems_event_set event_to_send;
81
82    return_status = rtems_task_ident (RTEMS_SELF,
83                                      RTEMS_SEARCH_ALL_NODES,
84                                      &task_id);
85    if (return_status != RTEMS_SUCCESSFUL)
86      {
87        rtems_semaphore_obtain (results_sem,
88                                RTEMS_WAIT,
89                                RTEMS_NO_TIMEOUT);
90        error = RTEMS_TASK_IDENT_ERROR;
91        show_test_results (error, return_status);
92        analyse_tasks_results ();
93        exit (-1);
```

```
 94      }
 95    while (count-- > 0)
 96      {
 97        event_task_id = consumers_info [consumers_index][system].task_id;
 98
 99        event_to_send = 1 << event;
100
101        return_status = rtems_event_send(event_task_id,
102                                         event_to_send);
103        if (return_status != RTEMS_SUCCESSFUL)
104          {
105            rtems_semaphore_obtain (results_sem,
106                                    RTEMS_WAIT,
107                                    RTEMS_NO_TIMEOUT);
108            set_producer_result (-1, task_id, system);
109            error = RTEMS_EVENT_SEND_ERROR;
110            show_test_results (error, return_status);
111            analyse_tasks_results ();
112            exit (-1);
113          }
114
115        consumers_info[consumers_index][system].events_sent[event]++;
116        /* Get next index (circular array) */
117        consumers_index = (++consumers_index) % NUMBER_OF_CONSUMERS;
118
119        rtems_task_wake_after (RTEMS_YIELD_PROCESSOR);
120        event++;
121        if (event == 32)
122          {
123            event = 0;
124          }
125      }
126    set_producer_result (0, task_id, system);
127  }
```

evt-consumer.c

```
  1  /*************************************************************************
  2   SRC-MODULE : Event Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-evt/evt-consumer.c,v $
  6   $Id: evt-consumer.c,v 1.1 2003/10/10 15:44:26 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Event
```

```
20   Manager. This file contains the consumer task for this workload.
21
22   CREATED ON : 09-10-2003
23   CHANGED ON : $Date: 2003/10/10 15:44:26 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: evt-consumer.c,v $
36   Revision 1.1  2003/10/10 15:44:26  lhenriques
37   First version of the stress-testing workload for the event manager.
38
39
40   ******************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-evt.h"
45
46  void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
47  {
48    int i;
49    for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
50      {
51        if (consumers_info[i][system].task_id == task_id)
52          {
53            consumers_info[i][system].task_id = -1;
54            return;
55          }
56      }
57       printf ("[ERROR] Could not obtain the consumer %d index in consumers
array.\n", task_id);
58  }
59
60  rtems_task consumer_task (rtems_task_argument consumer_arg)
61  {
62    /* Get system number index for the semaphore array */
63    rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
64    rtems_status_code return_status = RTEMS_NOT_DEFINED;
65    rtems_unsigned32 error = NO_ERROR;
66    rtems_id task_id;
67    rtems_event_set event_out;
68    consumer_info * this_consumer = NULL;
69    int evt = 0;
70    int i;
71
72    return_status = rtems_task_ident (RTEMS_SELF,
73                                      RTEMS_SEARCH_ALL_NODES,
74                                      &task_id);
75    if (return_status != RTEMS_SUCCESSFUL)
76      {
```

```
 77        rtems_semaphore_obtain (results_sem,
 78                                RTEMS_WAIT,
 79                                RTEMS_NO_TIMEOUT);
 80        error = RTEMS_TASK_IDENT_ERROR;
 81        show_test_results (error, return_status);
 82        analyse_tasks_results ();
 83        exit (-1);
 84      }
 85
 86    /* Get semaphore ID */
 87    for (i = 0; (i < NUMBER_OF_CONSUMERS) && (this_consumer == NULL); i++)
 88      {
 89        if (consumers_info[i][system].task_id == task_id)
 90          {
 91            this_consumer = &consumers_info[i][system];
 92          }
 93      }
 94    if (this_consumer == NULL)
 95      {
 96        rtems_semaphore_obtain (results_sem,
 97                                RTEMS_WAIT,
 98                                RTEMS_NO_TIMEOUT);
 99        error = RTEMS_GET_CONSUMER_INFO_ERROR;
100        show_test_results (error, 0);
101        analyse_tasks_results ();
102        exit (-1);
103      }
104
105    while (TRUE)
106      {
107        evt = 0;
108        return_status = rtems_event_receive (RTEMS_ALL_EVENTS,
109                                             (RTEMS_WAIT | RTEMS_EVENT_ANY),
110                                             RTEMS_NO_TIMEOUT,
111                                             &event_out);
112        if (return_status != RTEMS_SUCCESSFUL)
113          {
114            rtems_semaphore_obtain (results_sem,
115                                    RTEMS_WAIT,
116                                    RTEMS_NO_TIMEOUT);
117            error = RTEMS_EVENT_RECEIVE_ERROR;
118            show_test_results (error, return_status);
119            analyse_tasks_results ();
120            exit (-1);
121          }
122
123        /* Check which events have arrived */
124        while (event_out != 0)
125          {
126            if (event_out & 0x00000001)
127              {
128                this_consumer->events_received[evt]++;
129              }
130            event_out = event_out >> 1;
131            evt++;
132          }
133      }
134  }
```

B.7. Partition Manager Workload

rtems-cmp-cl-prt.h

```
/*****************************************************************************
 SRC-MODULE : Partition Manager Workload
 MODULE-VERS : N/A

          $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-testing/implementation/classic-workloads/rtems-cmp-cl-prt/rtems-cmp-cl-prt.h,v $
 $Id: rtems-cmp-cl-prt.h,v 1.2 2003/10/24 13:19:52 lhenriques Exp $
 $State: Exp $
 $Locker:  $

 Copyright (c) Critical Software (www.criticalsoftware.com)

 SPEC-NO : CSW-RAMS-2003-RPT-1335

 OS-TYPE : RTEMS 4.5.0

 AUTHOR : lhenriques

 KEYWORDS : ----
 PURPOSE : Stress the RTEMS Classical API for the Partition Manager.

 CREATED ON : 09-10-2003
 CHANGED ON : $Date: 2003/10/24 13:19:52 $
 CHANGED BY : $Author: lhenriques $

 $Revision: 1.2 $
 STICKY TAG : $Name:  $

 INSPECTED ON:
 MODERATOR :

 TABLES : none.

 HISTORY
 $Log: rtems-cmp-cl-prt.h,v $
 Revision 1.2  2003/10/24 13:19:52  lhenriques
 Changed default parameters.

 Revision 1.1  2003/10/10 15:43:08  lhenriques
 First version of the stress-testing workload for the partition manager.


 *****************************************************************************/

/*
   * The following definitions are the parameters that shall be changed to generated
 * diferent loads on the target.
 */

/* Number of producer tasks */
#define NUMBER_OF_PRODUCERS 64

```

```
 52  /* Stack size for the producers tasks */
 53  #define PRODUCERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
 54
 55  /* Producers priority */
 56  #define PRODUCERS_PRIORITY 3
 57
 58  /* Producers tasks mode */
 59  #define PRODUCERS_TASK_MODE RTEMS_DEFAULT_MODES
 60
 61  /* Producers tasks attributes */
 62  #define PRODUCERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 63
 64  /* Number of consumers tasks */
 65  #define NUMBER_OF_CONSUMERS 64
 66
 67  /* Stack size for the consumers tasks */
 68  #define CONSUMERS_TASK_STACK_SIZE RTEMS_MINIMUM_STACK_SIZE
 69
 70  /* Consumers tasks mode */
 71  #define CONSUMERS_TASK_MODE RTEMS_DEFAULT_MODES
 72
 73  /* Consumers tasks attributes */
 74  #define CONSUMERS_TASK_ATTR RTEMS_DEFAULT_ATTRIBUTES
 75
 76  /* Consumers priority */
 77  #define CONSUMERS_PRIORITY 2
 78
 79  /* Number of systems (producers/consumers systems) */
 80  #define NUMBER_OF_SYSTEMS 4
 81
 82  /* Number of ticks to wait until workload finishes */
 83  #define TEST_TIMEOUT 20000
 84
 85  /*************************************************************************/
 86  /* The following definitions are workload dependent.                     */
 87  /*************************************************************************/
 88
 89  /* The size of each partition */
 90  #define PARTITION_SIZE 1024
 91
 92  /* The size of each partition buffer */
 93  #define PARTITION_BUFFER_SIZE 16
 94
 95  /* Number of buffers to obtain by each producer */
 96  #define NUMBER_OF_BUFFERS 100
 97
 98  #define MESSAGE_QUEUE_ATTRIBUTES (RTEMS_FIFO | RTEMS_LOCAL)
 99
100  /* Time to wait before producing another item */
101  #define DELAY_BETWEEN_ITEMS 0
102
103  /*
104   * Enumeration with all the possible errors.
105   */
106  enum {
107    NO_ERROR,
108    RTEMS_SEMAPHORE_CREATE_ERROR,
109    RTEMS_MESSAGE_QUEUE_CREATE_ERROR,
```

```
110     RTEMS_SEMAPHORE_OBTAIN_ERROR,
111     RTEMS_TASK_CREATE_ERROR,
112     RTEMS_TASK_START_ERROR,
113     RTEMS_TASK_WAKE_AFTER_ERROR,
114     RTEMS_TASK_IDENT_ERROR,
115     RTEMS_MESSAGE_QUEUE_SEND_ERROR,
116     RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR,
117     RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR,
118     RTEMS_PARTITION_CREATE_ERROR,
119     RTEMS_PARTITION_GET_BUFFER_ERROR,
120     RTEMS_PARTITION_WRONG_TASK_ID_ERROR,
121     RTEMS_PARTITION_RETURN_BUFFER_ERROR
122   };
123
124   typedef struct {
125     rtems_id task_id;
126     rtems_id * buffer;
127   } message;
128
129   /* The maximum size of a message */
130   #define MAX_MESSAGE_SIZE (sizeof (message))
131
132   /* Arrays to store the producers and consumers IDs */
133   extern rtems_id producers_array[NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
134   extern rtems_id consumers_array[NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
135
136   /* Array containing the message queues for all systems */
137   extern rtems_id msg_queues_array[NUMBER_OF_SYSTEMS];
138
139   /* Array containing the partitions for all systems */
140           extern     rtems_unsigned32     partitions_array     [NUMBER_OF_SYSTEMS]
CPU_STRUCTURE_ALIGNMENT;
141
142   /* Array containing the IDs of every partition */
143   extern rtems_id partitions_ids_array [NUMBER_OF_SYSTEMS];
144
145   /* Semaphore that shall be obtained in order to exit the workload */
146   extern rtems_id results_sem;
147
148   /*
149    * These are the producer and consumer tasks.
150    */
151   rtems_task producer_task (rtems_task_argument producer_arg);
152   rtems_task consumer_task (rtems_task_argument consumer_arg);
153
154   /*
155    * This function shall analyse the results in each task,
156    */
157   void analyse_tasks_results ();
158
159   /*
160    * This function simply checks the errors in the test
161   H */
162        void    show_test_results    (rtems_unsigned32    error,    rtems_status_code
return_status);
163
164   void parse_error (rtems_unsigned32 error, rtems_status_code return_status);
165
```

```
166  /*
167   * This function switchs the return status parameter and prints a string with
168   * the corresponding error code.
169   */
170  void show_error_code (rtems_status_code return_status);
```

rtems-cmp-cl-prt.c

```
  1  /*****************************************************************************
  2   SRC-MODULE : Partition Manager Workload
  3   MODULE-VERS : N/A
  4
  5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-prt/rtems-cmp-cl-prt.c,v $
  6   $Id: rtems-cmp-cl-prt.c,v 1.2 2003/10/24 13:19:41 lhenriques Exp $
  7   $State: Exp $
  8   $Locker:  $
  9
 10   Copyright (c) Critical Software (www.criticalsoftware.com)
 11
 12   SPEC-NO : CSW-RAMS-2003-RPT-1335
 13
 14   OS-TYPE : RTEMS 4.5.0
 15
 16   AUTHOR : lhenriques
 17
 18   KEYWORDS : ----
 19   PURPOSE : Stress the RTEMS Classical API for the Partition Manager.
 20
 21   CREATED ON : 09-10-2003
 22   CHANGED ON : $Date: 2003/10/24 13:19:41 $
 23   CHANGED BY : $Author: lhenriques $
 24
 25   $Revision: 1.2 $
 26   STICKY TAG : $Name:  $
 27
 28   INSPECTED ON:
 29   MODERATOR :
 30
 31   TABLES : none.
 32
 33   HISTORY
 34   $Log: rtems-cmp-cl-prt.c,v $
 35   Revision 1.2  2003/10/24 13:19:41  lhenriques
 36   Corrected some problems in the tasks analysis.
 37
 38   Revision 1.1  2003/10/10 15:43:08  lhenriques
 39   First version of the stress-testing workload for the partition manager.
 40
 41
 42   *****************************************************************************/
 43
 44  #include <bsp.h>
 45  #include <stdio.h>
 46  #include <stdlib.h>
 47  #include "rtems-cmp-cl-prt.h"
 48
```

```
 49 rtems_id producers_array [NUMBER_OF_PRODUCERS][NUMBER_OF_SYSTEMS];
 50 rtems_id consumers_array [NUMBER_OF_CONSUMERS][NUMBER_OF_SYSTEMS];
 51
 52 rtems_id msg_queues_array [NUMBER_OF_SYSTEMS];
 53
 54 rtems_id partitions_ids_array [NUMBER_OF_SYSTEMS];
 55 rtems_unsigned32 partitions_array [NUMBER_OF_SYSTEMS] CPU_STRUCTURE_ALIGNMENT;
 56
 57 rtems_id results_sem;
 58
 59 int created_producers = 0;
 60 int created_consumers = 0;
 61
 62 /*
 63  * Fatal error handler. It is used to find out whether an halt has
 64  * occurred during the workload execution.
 65  */
 66 rtems_extension fatal_error_handler (rtems_unsigned32 the_source,
 67                                      boolean is_internal,
 68                                      rtems_unsigned32 the_error)
 69 {
 70   printf ("A Fatal error has occurred!\n");
 71   printf ("Source: ");
 72   /* Find source */
 73   switch (the_source)
 74     {
 75     case INTERNAL_ERROR_CORE:
 76       printf ("INTERNAL_ERROR_CORE");
 77       break;
 78     case INTERNAL_ERROR_RTEMS_API:
 79       printf ("INTERNAL_ERROR_RTEMS_API");
 80       break;
 81     case INTERNAL_ERROR_POSIX_API:
 82       printf ("INTERNAL_ERROR_POSIX_API");
 83       break;
 84     case INTERNAL_ERROR_ITRON_API:
 85       printf ("INTERNAL_ERROR_ITRON_API");
 86     default:
 87       printf ("UNKOWN (%d)", the_source);
 88     }
 89   printf ("\n");
 90
 91   if (is_internal == TRUE)
 92     {
 93       printf ("It is an INTERNAL error.\n");
 94
 95       printf ("Error: ");
 96       /* Find the error itself */
 97       switch (the_error)
 98         {
 99         case INTERNAL_ERROR_NO_CONFIGURATION_TABLE:
100           printf ("INTERNAL_ERROR_NO_CONFIGURATION_TABLE");
101           break;
102         case INTERNAL_ERROR_NO_CPU_TABLE:
103           printf ("INTERNAL_ERROR_NO_CPU_TABLE");
104           break;
105         case INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS:
106           printf ("INTERNAL_ERROR_INVALID_WORKSPACE_ADDRESS");
```

```
107           break;
108         case INTERNAL_ERROR_TOO_LITTLE_WORKSPACE:
109           printf ("INTERNAL_ERROR_TOO_LITTLE_WORKSPACE");
110           break;
111         case INTERNAL_ERROR_WORKSPACE_ALLOCATION:
112           printf ("INTERNAL_ERROR_WORKSPACE_ALLOCATION");
113           break;
114         case INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL:
115           printf ("INTERNAL_ERROR_INTERRUPT_STACK_TOO_SMALL");
116           break;
117         case INTERNAL_ERROR_THREAD_EXITTED:
118           printf ("INTERNAL_ERROR_THREAD_EXITTED");
119           break;
120         case INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION:
121           printf ("INTERNAL_ERROR_INCONSISTENT_MP_INFORMATION");
122           break;
123         case INTERNAL_ERROR_INVALID_NODE:
124           printf ("INTERNAL_ERROR_INVALID_NODE");
125           break;
126         case INTERNAL_ERROR_NO_MPCI:
127           printf ("INTERNAL_ERROR_NO_MPCI");
128           break;
129         case INTERNAL_ERROR_BAD_PACKET:
130           printf ("INTERNAL_ERROR_BAD_PACKET");
131           break;
132         case INTERNAL_ERROR_OUT_OF_PACKETS:
133           printf ("INTERNAL_ERROR_OUT_OF_PACKETS");
134           break;
135         case INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS:
136           printf ("INTERNAL_ERROR_OUT_OF_GLOBAL_OBJECTS");
137           break;
138         case INTERNAL_ERROR_OUT_OF_PROXIES:
139           printf ("INTERNAL_ERROR_OUT_OF_PROXIES");
140           break;
141         case INTERNAL_ERROR_INVALID_GLOBAL_ID:
142           printf ("INTERNAL_ERROR_INVALID_GLOBAL_ID");
143           break;
144         case INTERNAL_ERROR_BAD_STACK_HOOK:
145           printf ("INTERNAL_ERROR_BAD_STACK_HOOK");
146           break;
147         case INTERNAL_ERROR_BAD_ATTRIBUTES:
148           printf ("INTERNAL_ERROR_BAD_ATTRIBUTES");
149           break;
150         default:
151           printf ("UNKNOWN (%d)", the_error);
152         }
153       printf ("\n");
154
155     }
156   else
157     {
158       printf ("It is NOT an internal error.\n");
159       /* Assume an RTEMS Classic API error... */
160       show_error_code (the_error);
161     }
162
163   analyse_tasks_results ();
164   exit (-1);
```

```
165  }
166
167  /*
168   * This structure defines the user extensions entry points
169   */
170  rtems_extensions_table user_extensions =
171    {
172      NULL,               /* task creation extension */
173      NULL,               /* task start extension */
174      NULL,               /* task restart extension */
175      NULL,               /* task delete extension */
176      NULL,               /* task switch extension */
177      NULL,               /* task begin extension */
178      NULL,               /* task exitted extension */
179      fatal_error_handler /* fatal error extension */
180    };
181
182
183  /*
184   * Init task is the first to be executed.
185   */
186  rtems_task Init(rtems_task_argument ignored)
187  {
188    rtems_unsigned32 error = NO_ERROR;
189    rtems_status_code return_status = RTEMS_NOT_DEFINED;
190    rtems_unsigned32 system_index;
191    rtems_unsigned32 index;
192    rtems_unsigned32 max_index;
193    rtems_name results_sem_name;
194    rtems_name table_name;
195    rtems_id table_id;
196
197    /* Add user extension table to handle fata error */
198    table_name = rtems_build_name ('U', 'S', 'E', 'R');
199    return_status = rtems_extension_create (table_name,
200                                            &user_extensions,
201                                            &table_id);
202
203    if (return_status != RTEMS_SUCCESSFUL)
204      {
205              printf ("Error registering an user handler to the fatal error
extension.\n");
206        show_error_code (return_status);
207        exit (-1);
208      }
209
210    /* Create the results semaphore */
211    results_sem_name = rtems_build_name ('E', 'X', 'I', 'T');
212    return_status = rtems_semaphore_create (results_sem_name,
213                                            1, /* Only one task can exit */
214                                            RTEMS_DEFAULT_ATTRIBUTES,
215                                            RTEMS_NO_PRIORITY,
216                                            &results_sem);
217    /* If this fails... exit */
218    if (return_status != RTEMS_SUCCESSFUL)
219      {
220        printf ("Error creating semaphore for exit point.\n");
221        show_error_code (return_status);
```

```
222         exit (-1);
223       }
224
225     /* Create all the NUMBER_OF_SYSTEMS systems. */
226     for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
227       {
228         rtems_name producer_task_name;
229         rtems_name consumer_task_name;
230         rtems_name msg_queue_name;
231         rtems_name partition_name;
232
233         /* Create the resource:
234          *  - A message queue for each system
235          *  - A Partition
236          */
237         msg_queue_name = system_index + 1;
238         return_status = rtems_message_queue_create (msg_queue_name,
239                                                     (PARTITION_SIZE /
PARTITION_BUFFER_SIZE + 1),
240                                                     MAX_MESSAGE_SIZE,
241                                                     MESSAGE_QUEUE_ATTRIBUTES,
242
&msg_queues_array[system_index]);
243         if (return_status != RTEMS_SUCCESSFUL)
244           {
245             rtems_semaphore_obtain (results_sem,
246                                     RTEMS_WAIT,
247                                     RTEMS_NO_TIMEOUT);
248             error = RTEMS_MESSAGE_QUEUE_CREATE_ERROR;
249             show_test_results (error, return_status);
250             analyse_tasks_results ();
251             exit (-1);
252           }
253
254         partitions_array[system_index] = malloc (PARTITION_SIZE);
255         partition_name = system_index + 1;
256         return_status = rtems_partition_create (partition_name,
257                                                 (void *)partitions_array
[system_index],
258                                                 PARTITION_SIZE,
259                                                 PARTITION_BUFFER_SIZE,
260                                                 RTEMS_DEFAULT_ATTRIBUTES,
261
&partitions_ids_array[system_index]);
262         if (return_status != RTEMS_SUCCESSFUL)
263           {
264             rtems_semaphore_obtain (results_sem,
265                                     RTEMS_WAIT,
266                                     RTEMS_NO_TIMEOUT);
267             error = RTEMS_PARTITION_CREATE_ERROR;
268             show_test_results (error, return_status);
269             analyse_tasks_results ();
270             exit (-1);
271           }
272
273         /* Create all the producers and consumers for this system */
274                   max_index   =   (NUMBER_OF_PRODUCERS   <=   NUMBER_OF_CONSUMERS   ?
NUMBER_OF_CONSUMERS : NUMBER_OF_PRODUCERS);
```

```
275         for (index = 0; index < max_index; index++)
276           {
277             if (index < NUMBER_OF_PRODUCERS)
278               {
279                 /* Create a producer */
280                 producer_task_name = index + 1;
281                 return_status = rtems_task_create (producer_task_name,
282                                                    PRODUCERS_PRIORITY,
283                                                    PRODUCERS_TASK_STACK_SIZE,
284                                                    PRODUCERS_TASK_MODE,
285                                                    PRODUCERS_TASK_ATTR,
286
&producers_array[index][system_index]);
287                 if (return_status != RTEMS_SUCCESSFUL)
288                   {
289                     rtems_semaphore_obtain (results_sem,
290                                             RTEMS_WAIT,
291                                             RTEMS_NO_TIMEOUT);
292                     error = RTEMS_TASK_CREATE_ERROR;
293                     show_test_results (error, return_status);
294                     analyse_tasks_results ();
295                     exit (-1);
296                   }
297
298                                                  return_status   =   rtems_task_start
(producers_array[index][system_index],
299                                                     producer_task,
300                                                     system_index);
301                 if (return_status != RTEMS_SUCCESSFUL)
302                   {
303                     rtems_semaphore_obtain (results_sem,
304                                             RTEMS_WAIT,
305                                             RTEMS_NO_TIMEOUT);
306                     error = RTEMS_TASK_START_ERROR;
307                     show_test_results (error, return_status);
308                     analyse_tasks_results ();
309                     exit (-1);
310                   }
311                 created_producers++;
312               }
313             if (index < NUMBER_OF_CONSUMERS)
314               {
315                 /* Create a consumer */
316                 consumer_task_name = index + 1;
317                 return_status = rtems_task_create (consumer_task_name,
318                                                    CONSUMERS_PRIORITY,
319                                                    CONSUMERS_TASK_STACK_SIZE,
320                                                    CONSUMERS_TASK_MODE,
321                                                    CONSUMERS_TASK_ATTR,
322
&consumers_array[index][system_index]);
323                 if (return_status != RTEMS_SUCCESSFUL)
324                   {
325                     rtems_semaphore_obtain (results_sem,
326                                             RTEMS_WAIT,
327                                             RTEMS_NO_TIMEOUT);
328                     error = RTEMS_TASK_CREATE_ERROR;
329                     show_test_results (error, return_status);
```

```
330                   analyse_tasks_results ();
331                   exit (-1);
332                 }
333
334                                                 return_status   =   rtems_task_start
(consumers_array[index][system_index],
335                                                    consumer_task,
336                                                    system_index);
337               if (return_status != RTEMS_SUCCESSFUL)
338                 {
339                   rtems_semaphore_obtain (results_sem,
340                                           RTEMS_WAIT,
341                                           RTEMS_NO_TIMEOUT);
342                   error = RTEMS_TASK_START_ERROR;
343                   show_test_results (error, return_status);
344                   analyse_tasks_results ();
345                   exit (-1);
346                 }
347               created_consumers++;
348             }
349         }
350     }
351
352   /* Wait for all tasks or for an error. */
353   return_status = rtems_task_wake_after (TEST_TIMEOUT);
354
355   if (return_status != RTEMS_SUCCESSFUL)
356     {
357       error = RTEMS_TASK_WAKE_AFTER_ERROR;
358     }
359
360   rtems_semaphore_obtain (results_sem,
361                           RTEMS_WAIT,
362                           RTEMS_NO_TIMEOUT);
363   show_test_results (error, return_status);
364   analyse_tasks_results ();
365
366   exit (0);
367 }
368
369 void show_test_results (rtems_unsigned32 error, rtems_status_code return_status)
370 {
371   printf ("================================================\n");
372   printf ("Test Paramenters\n");
373   printf ("================================================\n");
374   printf ("Number of producers: %d\n", NUMBER_OF_PRODUCERS);
375   printf ("Producers task stack size: %d\n", PRODUCERS_TASK_STACK_SIZE);
376   printf ("Producers priority: %d\n", PRODUCERS_PRIORITY);
377   printf ("Producers task mode: %d\n", PRODUCERS_TASK_MODE);
378   printf ("Producers task attributes: %d\n", PRODUCERS_TASK_ATTR);
379   printf ("Number of consumers: %d\n", NUMBER_OF_CONSUMERS);
380   printf ("Consumers task stack size: %d\n", CONSUMERS_TASK_STACK_SIZE);
381   printf ("Consumers task mode: %d\n", CONSUMERS_TASK_MODE);
382   printf ("Consumers task attributes: %d\n", CONSUMERS_TASK_ATTR);
383   printf ("Consumers priority: %d\n", CONSUMERS_PRIORITY);
384   printf ("Number of producers/consumers systems: %d\n", NUMBER_OF_SYSTEMS);
385   printf ("================================================\n");
386   /* Workload specific parameters */
```

```
387    printf ("Size of partitions: %d\n", PARTITION_SIZE);
388    printf ("Size of partitions buffers: %d\n", PARTITION_BUFFER_SIZE);
389    printf ("Number of buffers by producer: %d\n", NUMBER_OF_BUFFERS);
390
391    if (error == NO_ERROR)
392      {
393        printf ("Systems created sucessfully:\n");
394      }
395    else
396      {
397        parse_error (error, return_status);
398      }
399  }
400
401  void analyse_tasks_results ()
402  {
403    int system_index, index;
404    int number_of_consumers = 0;
405    int number_of_producers = 0;
406    int failed_consumers = 0;
407    int failed_consumer_index = -1;
408    int failed_consumer_system = -1;
409    int failed_producers = 0;
410    int failed_producer_index = -1;
411    int failed_producer_system = -1;
412
413    for (system_index = 0; system_index < NUMBER_OF_SYSTEMS; system_index++)
414      {
415        /* Analyse producers array */
416        for (index = 0; index < NUMBER_OF_PRODUCERS; index++)
417          {
418            if (producers_array[index][system_index] != 0)
419              {
420                number_of_producers++;
421                     if (rtems_get_class (producers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
422                  {
423                    failed_producers++;
424                    if (failed_producer_index == -1)
425                      {
426                        failed_producer_index = index;
427                        failed_producer_system = system_index;
428                      }
429                  }
430              }
431          }
432        /* Analyse consumers array */
433        for (index = 0; index < NUMBER_OF_CONSUMERS; index++)
434          {
435            if (consumers_array[index][system_index] != 0)
436              {
437                number_of_consumers++;
438                     if (rtems_get_class (consumers_array[index][system_index]) !=
OBJECTS_RTEMS_TASKS)
439                  {
440                    failed_consumers++;
441                    if (failed_consumer_index == -1)
442                      {
```

```
443                         failed_consumer_index = index;
444                         failed_consumer_system = system_index;
445                       }
446                   }
447               }
448           }
449       }
450           printf ("Total number of successfully created producers: %d\n",
number_of_producers);
451     printf ("Number of failed producers: %d\n", failed_producers);
452     if (failed_producers > 0)
453       {
454               printf ("System of first failed producer (zero indexed): %d\n",
failed_producer_system);
455                printf ("Index of first failed producer (zero indexed): %d\n",
failed_producer_index);
456       }
457           printf ("Total number of successfully created consumers: %d\n",
number_of_consumers);
458     printf ("Number of failed consumers: %d\n", failed_consumers);
459     if (failed_consumers > 0)
460       {
461               printf ("System of first failed consumer (zero indexed): %d\n",
failed_consumer_system);
462                printf ("Index of first failed consumer (zero indexed): %d\n",
failed_consumer_index);
463       }
464  }
465
466  void parse_error (rtems_unsigned32 error, rtems_status_code return_status)
467  {
468         switch (error)
469           {
470           case RTEMS_SEMAPHORE_CREATE_ERROR:
471             printf ("Error creating semaphore.\n");
472             break;
473           case RTEMS_MESSAGE_QUEUE_CREATE_ERROR:
474             printf ("Error creating message queue.\n");
475             break;
476           case RTEMS_SEMAPHORE_OBTAIN_ERROR:
477             printf ("Error obtaining a semaphore.\n");
478             break;
479           case RTEMS_TASK_CREATE_ERROR:
480             printf ("Error creating task.\n");
481             break;
482           case RTEMS_TASK_START_ERROR:
483             printf ("Error starting task.\n");
484             break;
485           case RTEMS_TASK_WAKE_AFTER_ERROR:
486             printf ("Error waiting for tasks to finish.\n");
487             break;
488           case RTEMS_TASK_IDENT_ERROR:
489             printf ("Error obtaining thread ID.\n");
490             break;
491           case RTEMS_MESSAGE_QUEUE_SEND_ERROR:
492             printf ("Error sending message.\n");
493             break;
494           case RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR:
```

```
495           printf ("Error receiving message.\n");
496           break;
497         case RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR:
498                   printf ("Error receiving message: size is different from
expected.\n");
499           break;
500         case RTEMS_PARTITION_CREATE_ERROR:
501           printf ("Error creating partition.\n");
502           break;
503         case RTEMS_PARTITION_GET_BUFFER_ERROR:
504           printf ("Error getting buffer from partition.\n");
505           break;
506         case RTEMS_PARTITION_WRONG_TASK_ID_ERROR:
507           printf ("Wrong task ID in partition buffer.\n");
508           break;
509         case RTEMS_PARTITION_RETURN_BUFFER_ERROR:
510           printf ("Error return buffer to partition.\n");
511           break;
512         default:
513           printf ("Unknown failure. Possible bug in workload.\n");
514           break;
515         }
516       show_error_code (return_status);
517 }
518
519 void show_error_code (rtems_status_code return_status)
520 {
521   printf ("Error code: ");
522   switch (return_status)
523     {
524     case RTEMS_TASK_EXITTED:
525       printf ("RTEMS_TASK_EXITTED");
526       break;
527     case RTEMS_MP_NOT_CONFIGURED:
528       printf ("RTEMS_MP_NOT_CONFIGURED");
529       break;
530     case RTEMS_INVALID_NAME:
531       printf ("RTEMS_INVALID_NAME");
532       break;
533     case RTEMS_INVALID_ID:
534       printf ("RTEMS_INVALID_ID");
535       break;
536     case RTEMS_TOO_MANY:
537       printf ("RTEMS_TOO_MANY");
538       break;
539     case RTEMS_TIMEOUT:
540       printf ("RTEMS_TIMEOUT");
541       break;
542     case RTEMS_OBJECT_WAS_DELETED:
543       printf ("RTEMS_OBJECT_WAS_DELETED");
544       break;
545     case RTEMS_INVALID_SIZE:
546       printf ("RTEMS_INVALID_SIZE");
547       break;
548     case RTEMS_INVALID_ADDRESS:
549       printf ("RTEMS_INVALID_ADDRESS");
550       break;
551     case RTEMS_INVALID_NUMBER:
```

```
552       printf ("RTEMS_INVALID_NUMBER");
553       break;
554     case RTEMS_NOT_DEFINED:
555       printf ("RTEMS_NOT_DEFINED");
556       break;
557     case RTEMS_RESOURCE_IN_USE:
558       printf ("RTEMS_RESOURCE_IN_USE");
559       break;
560     case RTEMS_UNSATISFIED:
561       printf ("RTEMS_UNSATISFIED");
562       break;
563     case RTEMS_INCORRECT_STATE:
564       printf ("RTEMS_INCORRECT_STATE");
565       break;
566     case RTEMS_ALREADY_SUSPENDED:
567       printf ("RTEMS_ALREADY_SUSPENDED");
568       break;
569     case RTEMS_ILLEGAL_ON_SELF:
570       printf ("RTEMS_ILLEGAL_ON_SELF");
571       break;
572     case RTEMS_ILLEGAL_ON_REMOTE_OBJECT:
573       printf ("RTEMS_ILLEGAL_ON_REMOTE_OBJECT");
574       break;
575     case RTEMS_CALLED_FROM_ISR:
576       printf ("RTEMS_CALLED_FROM_ISR");
577       break;
578     case RTEMS_INVALID_PRIORITY:
579       printf ("RTEMS_INVALID_PRIORITY");
580       break;
581     case RTEMS_INVALID_CLOCK:
582       printf ("RTEMS_INVALID_CLOCK");
583       break;
584     case RTEMS_INVALID_NODE:
585       printf ("RTEMS_INVALID_NODE");
586       break;
587     case RTEMS_NOT_CONFIGURED:
588       printf ("RTEMS_NOT_CONFIGURED");
589       break;
590     case RTEMS_NOT_OWNER_OF_RESOURCE:
591       printf ("RTEMS_NOT_OWNER_OF_RESOURCE");
592       break;
593     case RTEMS_NOT_IMPLEMENTED:
594       printf ("RTEMS_NOT_IMPLEMENTED");
595       break;
596     case RTEMS_INTERNAL_ERROR:
597       printf ("RTEMS_INTERNAL_ERROR");
598       break;
599     case RTEMS_NO_MEMORY:
600       printf ("RTEMS_NO_MEMORY");
601       break;
602     case RTEMS_IO_ERROR:
603       printf ("RTEMS_IO_ERROR");
604       break;
605     case RTEMS_PROXY_BLOCKING:
606       printf ("RTEMS_PROXY_BLOCKING");
607       break;
608     default:
609       printf ("UNKNOWN ERROR CODE");
```

```
610       }
611     printf ("\n");
612   }
613
614   /* configuration information */
615
616   #define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
617   #define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
618
619   #define CONFIGURE_RTEMS_INIT_TASKS_TABLE
620   #define CONFIGURE_INIT_TASK_STACK_SIZE (RTEMS_MINIMUM_STACK_SIZE * 2)
621    #define CONFIGURE_MAXIMUM_TASKS (NUMBER_OF_SYSTEMS * (NUMBER_OF_PRODUCERS +
NUMBER_OF_CONSUMERS) + 1)
622
623   #define CONFIGURE_MAXIMUM_SEMAPHORES 1
624
625   #define CONFIGURE_MAXIMUM_MESSAGE_QUEUES NUMBER_OF_SYSTEMS
626   #define CONFIGURE_MAXIMUM_PARTITIONS NUMBER_OF_SYSTEMS
627
628   #define CONFIGURE_MAXIMUM_USER_EXTENSIONS 2
629
630   #define CONFIGURE_INIT
631
632   #include <confdefs.h>
633
634   /* end of file */
```

prt-producer.c

```
1   /*************************************************************************
2    SRC-MODULE : Partition Manager Workload
3    MODULE-VERS : N/A
4
5             $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-prt/prt-producer.c,v $
6    $Id: prt-producer.c,v 1.1 2003/10/10 15:43:08 lhenriques Exp $
7    $State: Exp $
8    $Locker:  $
9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Partition
20   Manager. This file contains the producer task for this workload.
21
22   CREATED ON : 09-10-2003
23   CHANGED ON : $Date: 2003/10/10 15:43:08 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
```

```
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: prt-producer.c,v $
36   Revision 1.1  2003/10/10 15:43:08  lhenriques
37   First version of the stress-testing workload for the partition manager.
38
39
40   ***************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-prt.h"
45
46
47  void set_producer_error (rtems_id task_id, rtems_unsigned32 system)
48  {
49    int i;
50    for (i = 0; i < NUMBER_OF_PRODUCERS; i++)
51      {
52        if (producers_array[i][system] == task_id)
53          {
54            producers_array[i][system] = -1;
55            return;
56          }
57      }
58       printf ("[ERROR] Could not obtain the producer %d index in producers
array.\n", task_id);
59  }
60
61  rtems_task producer_task (rtems_task_argument producer_arg)
62  {
63    /* Get system number index the message queues array */
64    rtems_unsigned32 system = (rtems_unsigned32) producer_arg;
65    /* Number of rtems_message_queue_send calls */
66    rtems_signed32 count = NUMBER_OF_BUFFERS;
67    /* Get the message queue */
68    rtems_id msg_queue_id = msg_queues_array [system];
69    /* Get the partition ID */
70    rtems_id partition_id = partitions_ids_array [system];
71
72    rtems_status_code return_status = RTEMS_NOT_DEFINED;
73    rtems_unsigned32 error = NO_ERROR;
74    rtems_id * partition_buffer;
75    message msg;
76    rtems_id task_id;
77
78    return_status = rtems_task_ident (RTEMS_SELF,
79                                      RTEMS_SEARCH_ALL_NODES,
80                                      &task_id);
81    if (return_status != RTEMS_SUCCESSFUL)
82      {
83        rtems_semaphore_obtain (results_sem,
84                                RTEMS_WAIT,
```

```
 85                                RTEMS_NO_TIMEOUT);
 86        error = RTEMS_TASK_IDENT_ERROR;
 87        show_test_results (error, return_status);
 88        analyse_tasks_results ();
 89        exit (-1);
 90      }
 91
 92    while (count-- > 0)
 93      {
 94        return_status = rtems_partition_get_buffer (partition_id,
 95                                                    (void *)&partition_buffer);
 96                if  ((return_status  !=  RTEMS_SUCCESSFUL)  &&  (return_status  !=
RTEMS_UNSATISFIED))
 97          {
 98            rtems_semaphore_obtain (results_sem,
 99                                    RTEMS_WAIT,
100                                    RTEMS_NO_TIMEOUT);
101            error = RTEMS_PARTITION_GET_BUFFER_ERROR;
102            show_test_results (error, return_status);
103            analyse_tasks_results ();
104            exit (-1);
105          }
106        if (return_status != RTEMS_UNSATISFIED)
107          {
108            partition_buffer[0] = task_id;
109            msg.task_id = task_id;
110            msg.buffer = partition_buffer;
111            return_status = rtems_message_queue_send (msg_queue_id,
112                                                      &msg,
113                                                      MAX_MESSAGE_SIZE);
114            if (return_status != RTEMS_SUCCESSFUL)
115              {
116                rtems_semaphore_obtain (results_sem,
117                                        RTEMS_WAIT,
118                                        RTEMS_NO_TIMEOUT);
119                set_producer_error (task_id, system);
120                error = RTEMS_MESSAGE_QUEUE_SEND_ERROR;
121                show_test_results (error, return_status);
122                analyse_tasks_results ();
123                exit (-1);
124              }
125          }
126
127        /* Wait before sending other message */
128        return_status = rtems_task_wake_after (RTEMS_YIELD_PROCESSOR);
129        if (return_status != RTEMS_SUCCESSFUL)
130          {
131            rtems_semaphore_obtain (results_sem,
132                                    RTEMS_WAIT,
133                                    RTEMS_NO_TIMEOUT);
134            set_producer_error (task_id, system);
135            error = RTEMS_TASK_WAKE_AFTER_ERROR;
136            show_test_results (error, return_status);
137            analyse_tasks_results ();
138            exit (-1);
139          }
140      }
141  }
```

prt-consumer.c

```
 1  /*****************************************************************************
 2   SRC-MODULE : Partition Manager Workload
 3   MODULE-VERS : N/A
 4
 5            $Source:    /home/cvscritical/esa/rams/services/rams02/services/stress-
testing/implementation/classic-workloads/rtems-cm
p-cl-prt/prt-consumer.c,v $
 6   $Id: prt-consumer.c,v 1.1 2003/10/10 15:43:08 lhenriques Exp $
 7   $State: Exp $
 8   $Locker:  $
 9
10   Copyright (c) Critical Software (www.criticalsoftware.com)
11
12   SPEC-NO : CSW-RAMS-2003-RPT-1335
13
14   OS-TYPE : RTEMS 4.5.0
15
16   AUTHOR : lhenriques
17
18   KEYWORDS : ----
19   PURPOSE : Stress the RTEMS Classical API for the Partition
20   Manager. This file contains the consumer task for this workload.
21
22   CREATED ON : 09-10-2003
23   CHANGED ON : $Date: 2003/10/10 15:43:08 $
24   CHANGED BY : $Author: lhenriques $
25
26   $Revision: 1.1 $
27   STICKY TAG : $Name:  $
28
29   INSPECTED ON:
30   MODERATOR :
31
32   TABLES : none.
33
34   HISTORY
35   $Log: prt-consumer.c,v $
36   Revision 1.1  2003/10/10 15:43:08  lhenriques
37   First version of the stress-testing workload for the partition manager.
38
39
40   *****************************************************************************/
41
42  #include <bsp.h>
43  #include <stdio.h>
44  #include "rtems-cmp-cl-prt.h"
45
46  void set_consumer_error (rtems_id task_id, rtems_unsigned32 system)
47  {
48    int i;
49    for (i = 0; i < NUMBER_OF_CONSUMERS; i++)
50      {
51        if (consumers_array[i][system] == task_id)
52          {
53            consumers_array[i][system] = -1;
```

```
54            return;
55          }
56      }
57       printf ("[ERROR] Could not obtain the consumer %d index in consumers
array.\n", task_id);
58  }
59
60  rtems_task consumer_task (rtems_task_argument consumer_arg)
61  {
62    /* Get system number index for the message queues array */
63    rtems_unsigned32 system = (rtems_unsigned32) consumer_arg;
64    /* Get the message queue */
65    rtems_id msg_queue_id = msg_queues_array[system];
66    /* Get partition ID */
67    rtems_id partition_id = partitions_ids_array[system];
68    rtems_status_code return_status = RTEMS_NOT_DEFINED;
69    rtems_unsigned32 error = NO_ERROR;
70    rtems_unsigned32 msg_size = 0;
71    message msg;
72    rtems_id task_id;
73
74    return_status = rtems_task_ident (RTEMS_SELF,
75                                      RTEMS_SEARCH_ALL_NODES,
76                                      &task_id);
77    if (return_status != RTEMS_SUCCESSFUL)
78      {
79        rtems_semaphore_obtain (results_sem,
80                                RTEMS_WAIT,
81                                RTEMS_NO_TIMEOUT);
82        error = RTEMS_TASK_IDENT_ERROR;
83        show_test_results (error, return_status);
84        analyse_tasks_results ();
85        exit (-1);
86      }
87    while (TRUE)
88      {
89        return_status = rtems_message_queue_receive (msg_queue_id,
90                                                     &msg,
91                                                     &msg_size,
92                                                     RTEMS_WAIT,
93                                                     RTEMS_NO_TIMEOUT);
94        if (return_status != RTEMS_SUCCESSFUL)
95          {
96            rtems_semaphore_obtain (results_sem,
97                                    RTEMS_WAIT,
98                                    RTEMS_NO_TIMEOUT);
99            error = RTEMS_MESSAGE_QUEUE_RECEIVE_ERROR;
100           set_consumer_error (task_id, system);
101           show_test_results (error, return_status);
102           analyse_tasks_results ();
103           exit (-1);
104         }
105       if (msg_size != MAX_MESSAGE_SIZE)
106         {
107           rtems_semaphore_obtain (results_sem,
108                                   RTEMS_WAIT,
109                                   RTEMS_NO_TIMEOUT);
110           error = RTEMS_MESSAGE_QUEUE_RECEIVE_WRONG_SIZE_ERROR;
```

```
111             set_consumer_error (task_id, system);
112             show_test_results (error, return_status);
113             analyse_tasks_results ();
114             exit (-1);
115           }
116
117         if (msg.buffer[0] != msg.task_id)
118           {
119             rtems_semaphore_obtain (results_sem,
120                                     RTEMS_WAIT,
121                                     RTEMS_NO_TIMEOUT);
122             error = RTEMS_PARTITION_WRONG_TASK_ID_ERROR;
123             set_consumer_error (task_id, system);
124             show_test_results (error, return_status);
125             analyse_tasks_results ();
126             exit (-1);
127           }
128
129         return_status = rtems_partition_return_buffer (partition_id,
130                                                        msg.buffer);
131         if (return_status != RTEMS_SUCCESSFUL)
132           {
133             rtems_semaphore_obtain (results_sem,
134                                     RTEMS_WAIT,
135                                     RTEMS_NO_TIMEOUT);
136             error = RTEMS_PARTITION_RETURN_BUFFER_ERROR;
137             set_consumer_error (task_id, system);
138             show_test_results (error, return_status);
139             analyse_tasks_results ();
140             exit (-1);
141           }
142       }
143   }
```