VXWORKS/X86 5.3.1

Disclaimer

Although all care has been taken to obtain correct information and accurate test results, Dedicated Systems Experts and Dedicated Systems Magazine cannot be liable for any incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report, even if Dedicated Systems Experts and Dedicated Systems Magazine have been advised of the possibility of such damages.

http://www.dedicated-systems.com

E-mail: info@dedicated-systems.com

EVALUATION REPORT LICENSE

This is a legal agreement between you (the downloader of this document) and/or your company and the company DEDICATED SYSTEMS EXPERTS NV, Gerechtsstraat 21, 1070 Brussels, Belgium.
It is not possible to download this document without registering and accepting this agreement on-line.

1. **GRANT**. Subject to the provisions contained herein, Dedicated Systems Experts hereby grants you a non-exclusive license to use its accompanying proprietary evaluation report for projects where you or your company are involved as major contractor or subcontractor. You are not entitled to support or telephone assistance in connection with this license.
2. **PRODUCT**. Dedicated Systems Experts shall furnish the evaluation report to you electronically via Internet. This license does not grant you any right to any enhancement or update to the document.
3. **TITLE**. Title, ownership rights, and intellectual property rights in and to the document shall remain in Dedicated Systems Experts and/or its suppliers or evaluated product manufacturers. The copyright laws of Belgium and all international copyright treaties protect the documents.
4. **CONTENT**. Title, ownership rights, and an intellectual property right in and to the content accessed through the document is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights to such content.
5. **YOU CAN NOT**:
   - You can not, make (or allow anyone else make) copies, whether digital, printed, photographic or others, except for backup reasons. The number of copies should be limited to 2. The copies should be exact replicates of the original (in paper or electronic format) with all copyright notices and logos.
   - You can not, place (or allow anyone else place) the evaluation report on an electronic board or other form of on line service without authorisation.
6. **INDEMNIFICATION**. You agree to indemnify and hold harmless Dedicated Systems Experts against any damages or liability of any kind arising from any use of this product other than the permitted uses specified in this agreement.
7. **DISCLAIMER OF WARRANTY**. All documents published by Dedicated Systems Experts on the World Wide Web Server or by any other means are provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. This disclaimer of warranty constitutes an essential part of the agreement.
8. **LIMITATION OF LIABILITY**. Neither Dedicated Systems Experts nor any of its directors, employees, partners or agents shall, under any circumstances, be liable to any person for any special, incidental, indirect or consequential damages, including, without limitation, damages resulting from use of OR RELIANCE ON the INFORMATION presented, loss of profits or revenues or costs of replacement goods, even if informed in advance of the possibility of such damages.
9. **ACCURACY OF INFORMATION**. Every effort has been made to ensure the accuracy of the information presented herein. However Dedicated Systems Experts assumes no responsibility for the accuracy of the information. Product information is subject to change without notice. Changes, if any, will be incorporated in new editions of these publications. Dedicated Systems Experts may make improvements and/or changes in the products and/or the programs described in these publications at any time without notice. Mention of non-Dedicated Systems Experts products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.
10. **JURISDICTION**. In case of any problems, the court of BRUSSELS-BELGIUM will have exclusive jurisdiction.

Agreed by downloading the document via the internet.

1 Summary

1.1 Product

VxWorks 5.3.1 from WindRiver Systems, Inc. (http://www.wrs.com)

Corporate headquarters: 500 Wind River Way. Alameda, CA 94501 Tel: (800) 545 WIND

1.2 Positive points

- Good development environment with an extensive set of tools.
- Extensive Application Program Interface.
- Predictable performance.
- Professional technical support.
- Support for a lot of different hardware platforms.

1.3 Negative points

- Manuals are not always clear.
- Does not qualify as a fault-tolerant, distributed system. Poor support for inter-processor communication.

1.4 Ratings

For a description of the ratings, the reader is referred to appendix D in the document “report definition and test plan”, which can be downloaded from the visitor center at http://www.dedicated-systems.com/encyc

Installation and Configuration	0	6	10
RTOS Architecture	0	6	10
API Richness	0	8	10
Internet support	0	8	10
Tools	0	9	10
Documentation and Support	0	7	10
Development methodology	0	8	10
Test Results	0	7	10

1.5 Pricing

A single development user license has a list price of $16,500. A single run-time license costs $1,000.

http://www.dedicated-systems.com
email: info@dedicated-systems.com

2 Introduction

This document contains two parts. Part I is the technical evaluation and part II is the practical evaluation. The technical evaluation is a theoretical study of the architectural and the design considerations of the OS, based on product documentation and experience. In the practical evaluation, various test results will be analyzed and commented.

This report broaches various technical issues that are discussed in the document “What makes a good RTOS?”. This document reflects the opinion of Real-time Consult on what is necessary in an OS to become a good RTOS. “What makes a good RTOS?” should therefore be considered an integral part of this evaluation report.

2.1 Different steps in creating such evaluation report

In the first step, the evaluated product is researched and submitted to a variety of tests. After processing and analyzing the test results, a first version of the report is drafted by Real-Time Consult. A copy of this first draft is then submitted for review to the vendor of the product.

During the second step, the comments and feedback of the vendor are studied. The comments, from which we feel that they are justified, are incorporated into the report.

2.2 Revision strategy

This report is revision 2.00. To improve the quality of these reports, the reader is encouraged to send us by email (info@dedicated-systems.com) or by using the FEEDBACK FORM on our web page, any suggestions, remarks and report inaccuracies that he/she might find useful to mention. The feedback from our readers will be considered and taken into account in the next minor revision. New minor revisions are available to the owners of the previous revision free of charge.

When new content is added to the report (e.g. extra tests) a major revision is created. New major revisions will be available for sale. Discounts will be offered to the owners of a previous revision.

3 Technical evaluation

In order to clarify the content of the following sections, the reader is referred to the document "report definition and test plan", which can be downloaded from the visitor center at http://www.dedicated-systems.com/encyc. This document defines (among other things) the different memory protection models, the objects and features mentioned in our API richness section. It also contains a detailed explanation of the tests performed.

3.1 Installation and Configuration

Installation and Configuration	0						6				10

Installation is pretty straightforward. Configuration is trickier. Due to the large variety of target architectures that are supported, the system is highly configurable. It might take an inexperienced user some time to setup host and target as desired.

3.1.1 Installation

Tornado 1.0.1 and VxWorks 5.3.1 come on a single CD. VxWorks is the dedicated RTOS running on the target, while Tornado, the complete development environment is running on the host.

During the first step of the installation, the user has to enter an installation key. By means of this key, the setup program determines which components the user is allowed to install. Setup then gives the opportunity to confirm the products by product name. The user can select which products from the list to install or not.

During the second step of the installation, a license manager needs to be installed. One license key is provided for every licensed product. There are 2 types of licenses, a *floating license* and a *single-user license*.

A *single-user license* is for one user on one machine. This is typical for a environment where there is no network and a floating license server is not available. Single-user licenses are node-locked, which means they can be used on only one host !

A *floating license* is for a number of users who have a network available. A single license server daemon grants licenses to a certain number of client applications. Licenses are distributed according to the number of seats purchased.

3.1.2 Configuration

Delivered together with Tornado are Board Support Packages (BSP). A BSP contains configuration files for a particular type of target. The user has a great deal of flexibility in setting up the host/target environment. The target can be booted from ROM or from the host via an ethernet network or other communications medium. PC based BSPs even allow the target to boot from floppy or hard drive. The default medium for host and target to communicate is via an ethernet network.

http://www.dedicated-systems.com
email: info@dedicated-systems.com

All this flexibility also implies that the user needs to have some basic knowledge of the system. An inexperienced user may need some time to figure out how to set up everything to his liking.

The vendor has informed us that the next version of Tornado, called Tornado II was designed to improve the “out of the box experience”. Tornado II was not used for this evaluation, as it was only formally released after the evaluation was finished.

3.2 RTOS Architecture

RTOS Architecture	0	6	10

VxWorks 5.3.1 is scalable and based on a variant of the client-server architecture. It is not message based. VxWorks does not qualify as a fault-tolerant distributed operating system, and has poor support for inter-processor communication in systems without shared memory.

3.2.1 System Architecture

The Wind River philosophy is to utilize two complementary and cooperating operating systems, and let each do what it does best. On one hand, there is the *host* machine which runs a general purpose operating system like Windows NT or UNIX, and is used for application development only. On the other hand, there is the *target* machine which runs VxWorks and handles the real-time chores. Only a small debug agent is resident on the target, all other development software runs on the host.

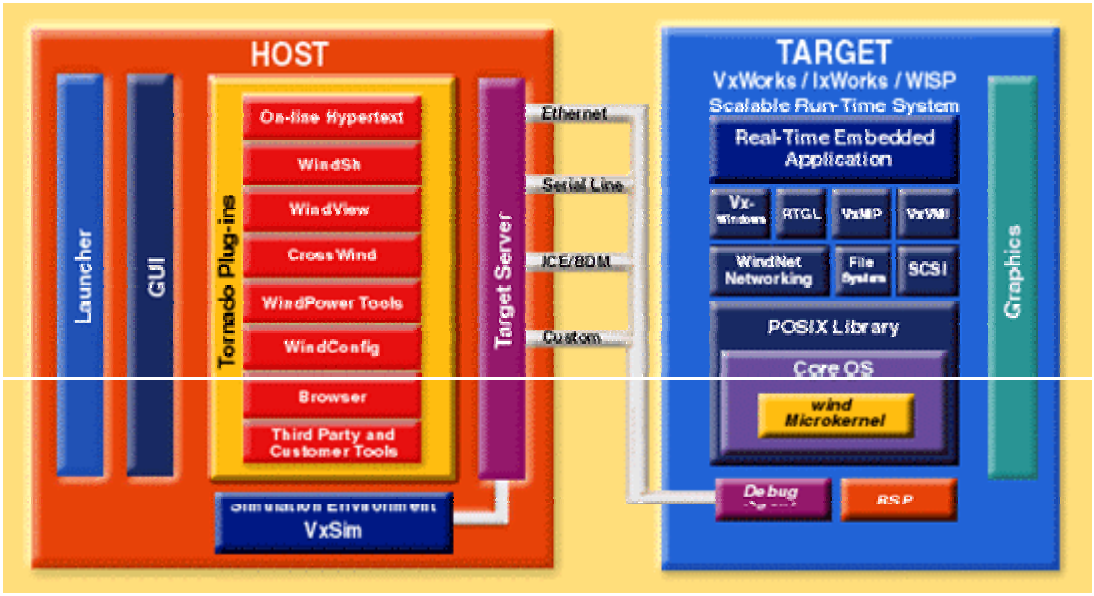

Figure 3.2-1 VxWorks – host and target machine (from [4])

VxWorks was initially a development and network environment for VRTX and pSOS. Only later on did Wind River Systems develop their own microkernel. The net results is that VxWorks was “client-server” from the beginning.

At the heart of the VxWorks run-time system is the *wind* microkernel. This microkernel supports a full range of real-time features including multi-tasking, scheduling, intertask synchronization/communication and memory management. All the other functionality is implemented as processes.

☹ No privilege protection is used, the privilege level is always 0 (supervisor mode), so there is no need for call gates. The GDT (Global Descriptor Table) has only 5 entries: a null and a reserved entry, an entry for

program code, an entry for program data and an entry for ISRs. This last entry is needed because ISRs in VxWorks by default use a separate stack (see 3.2.2.3).

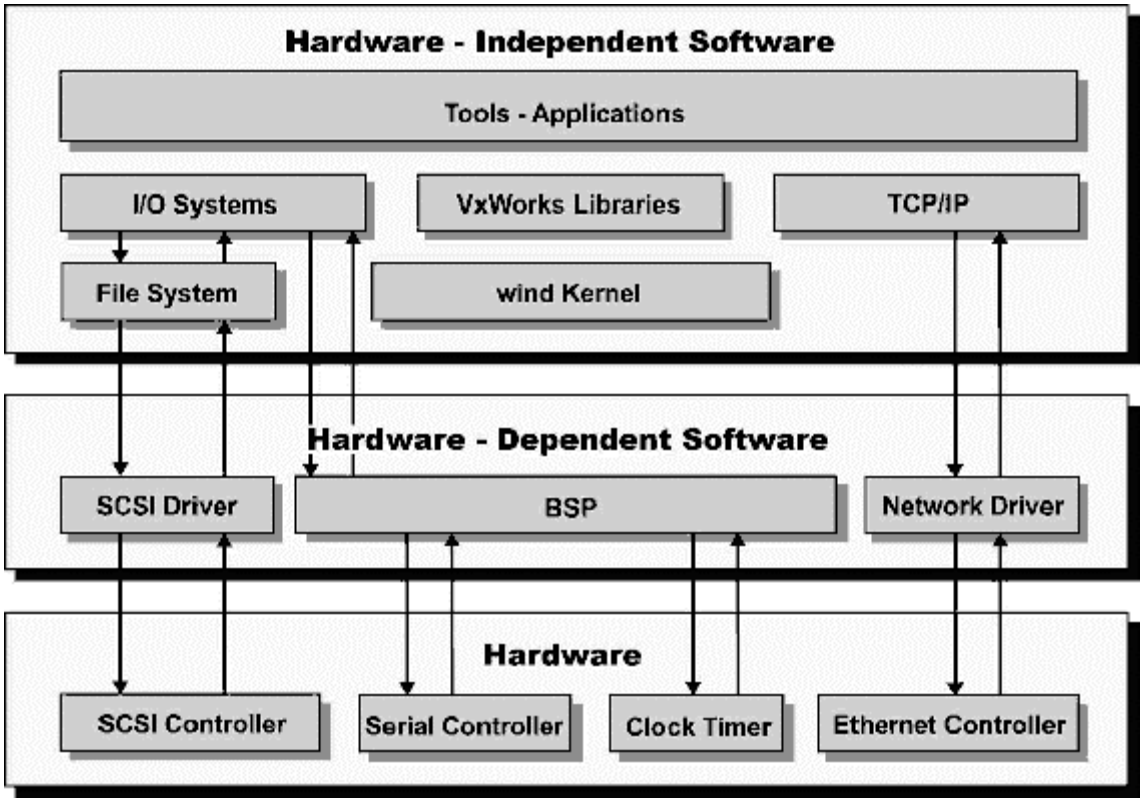

Figure 3.2-2: VxWorks system architecture (from [5])

☺ VxWorks 5.3.1 is very scalable. By including or excluding various modules, VxWorks can be configured for the use in small embedded system with tough memory constraints to complex systems where more functions are needed. Furthermore, individual modules themselves are scalable. Individual functions may be removed from the library, or specific kernel synchronization objects may be omitted if they are not required by the application.

☹ VxWorks has poor support for communication between processes located on different processors. Message queues for example can only be used to communicate if there is a shared memory area for these processors to access. Also, an optional component called “VxMP” needs to be purchased and installed to use such shared-memory objects.

Socket based communication can accomplish this without the use of shared memory (and is also location transparent to the application), but has a lot of drawbacks compared to conventional message queues (e.g. lack of data structure in socket based communication). Furthermore, the TCP/IP protocol introduces unnecessary overhead when the message data is being sent over a reliable communications media.

3.2.2 Basic System Facilities

3.2.2.1 Task Handling Method

The VxWorks 5.3.1 real-time kernel (wind) provides a basic multitasking environment. Each task has its own *context*, which is the CPU environment and system resources it sees when it is scheduled by the kernel for execution. On a context switch, the task's context is saved in something that is called a *task control block* (TCB).

VxWorks 5.3.1 offers both POSIX and a proprietary scheduling mechanism (wind scheduling). Different scheduling algorithms are available:

- *Preemptive priority scheduling*: this is the default algorithm in *wind*. This algorithm makes sure that the CPU is allocated to the highest priority thread ready to run. When a thread has a higher priority than the currently active thread, the high priority thread becomes ready to run immediately and the kernel immediately switches to the context of the high priority thread. For threads of equal priority, a FIFO scheduling mechanism is used.
- *Round-robin scheduling*: this scheduling algorithm is an extension of preemptive priority scheduling. It attempts to share the CPU more fairly among all ready threads of the same priority. The round-robin algorithm achieves this by assigning a time slice to threads of equal priority. A thread cannot execute longer than the duration of its time slice. When the time slice expires, the kernel gives other threads of equal priority a chance to execute for an equal time interval.

The differences between POSIX and wind scheduling is that wind scheduling applies the scheduling algorithm on a system wide basis, whereas POSIX scheduling algorithms are applied on a process-by-process basis. Of course, standard VxWorks doesn't have processes (only threads) unless an optional component is used (see section 3.2.2.2). Also, the POSIX priority numbering scheme is the inverse of the wind scheme. In POSIX, the higher the number, the higher the priority. In the wind scheme, it is the opposite: the lower the number, the higher the priority. Every thread can be assigned 1 of 256 priority levels.

VxWorks 5.3.1 has a rich set of task related features. For example: a task can protect itself from being deleted unexpectedly, a task can disable the scheduler so that it can't be preempted (interrupt handling will still work), hooks are provided which allow other routines to be activated when a task is created, deleted or a task context switch occurs, etc.

	VxWorks 5.3.1
Model	☹ Tasks containing single thread of execution. All tasks run in a single address space without any protection. 😐 VxVMI (optional component) allows each task to have its own address space.
Priority levels	☺ 256 levels.
Max. number of tasks	☺ Limited by the amount of memory available.
Scheduling policies	☺ Both POSIX and Wind scheduling. The *wind* scheduling policy is applied on a system-wide basis, while the POSIX scheduling policy is applied on a process-by-process basis. Preemptive priority.

	Round-robin.
Number of documented states	9

3.2.2.2 Memory Management Method

☹ In VxWorks 5.3.1, all systems and all application tasks share the same address space. This means that faulty applications could accidentally access system resources and compromise the stability of the entire system.

😐 However, WindRiver Systems does provide an additional component (VxVMI) that needs to be purchased separately and that allows every process to have its own private virtual memory. It is advisable to make use of this component during the development of applications. Having memory protection capabilities around during development and debugging can be very helpful to track down memory related bugs like stack or heap overflows.

VxWorks 5.3.1 (without VxVMI) does have some basic MMU support built in for protection with memory mapped devices. Details of the working of this mechanism depends on the hardware platform used. For the x86 processors for example, the user can configure the memory space into valid and invalid areas. If the processor tries to access addresses that are mapped as invalid, or addresses that are not mapped at all, a page fault will be generated.

When tasks share the same address space, they are actually threads. In order to qualify as a process, a task should have its own address space. According to the POSIX standard, a process is an address space with at least one thread executing in that address space and the system resources required for those threads.

	VxWorks 5.3.1
MMU	Not required, but supported (need extra component VxVMI).
Physical page size	8KB by default.
Paging/Swapping	Never.
Virtual memory	Supported (need extra component VxVMI).
Memory protection models	No protection: all tasks run in the same address space. User/user private VM: every user process has its own private virtual memory (need extra component VxVMI).

3.2.2.3 Interrupt Handling Method

To achieve the fastest possible response to external interrupts, interrupt service routines (ISRs) in VxWorks 5.3.1 run in a special context outside of any thread's context, so that there are no thread context switches involved.

Under normal circumstances, all ISRs use the same interrupt stack. This stack is allocated and initialized by the system at start-up according to specified configuration parameters. In order to avoid stack faults, the stack size must be large enough to handle the worst possible combination of nested interrupts.

Some architectures do not allow a separate interrupt stack. In this case, the ISRs use the stack of the interrupted thread. This means that threads need to be created with enough stack space to handle the worst possible combination of nested interrupts and the worst possible combination of ordinary nested calls.

An ISR does not run in a regular task context: it has no control block and all ISRs share the same stack. Because of this, there are some important limitations as to which facilities are available to ISRs. The basic restriction on ISRs is that they must not invoke routines that might cause the caller to block. For example, an ISR should not try to acquire a semaphore, because if the semaphore is not available, the kernel would try to switch the caller to the pended state. ISRs can release semaphores, thereby unblocking other threads that were pending on them.

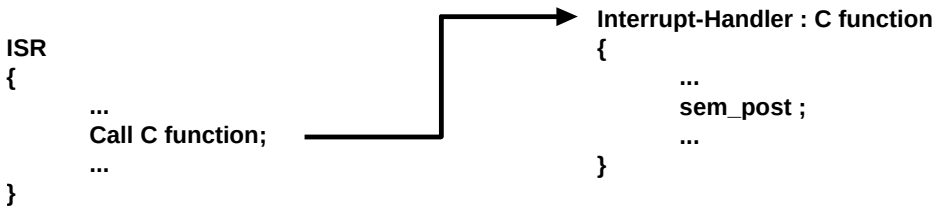

Figure 3.2-3 : Difference between ISR and user installed interrupt handler

It should be pointed out that the C function that the user attaches to a interrupt vector is not the actual ISR. Interrupts cannot directly vector to C functions. The ISR's address is stored in the interrupt vector table and is called directly from the hardware. The ISR performs some initial work (e.g. saving registers and setting up stack) and then calls the C function that was attached by the user. For this reason, we use the term interrupt handler (instead of ISR) to designate the user installed C handler function.

	VxWorks 5.3.1
Handling	☺ Nested and prioritized.
Context	Interrupt handlers run in a special context, outside any task's context.
Stack	Special interrupt stack. If the architecture doesn't allow a separate stack for ISRs, then the stack of the interrupted task is used.
Interrupt-to-task communication	Shared memory and ring buffers. Semaphores (release only). Message queues (send only) Pipes (write only)

http://www.dedicated-systems.com
email: info@dedicated-systems.com

	VxWorks 5.3.1
	Signals (send only)
Min. RAM	Less than 100 bytes.

3.3 API Richness

API Richness	0	8	10

Very extensive API, but it is missing conditional variables and event flags.

3.3.1 POSIX

VxWorks 5.3.1 supports many POSIX 1003.1b basic system calls, including process primitives, files and directories, I/O primitives, language services, and directory handling. In addition, VxWorks adheres to the final approved standard for POSIX 1003.1b Real-Time Extensions, including POSIX-compliant asynchronous I/O, counting semaphores, message queues, signals, memory management (page locking), and scheduling control.

3.3.2 Task Management

Thread management	YES
Get stack size	✓
Set stack size	✓
Get stack address	✓
Set stack address	✓
Get thread state	✓
Set thread state	✓
Get TCB	✓
Set TCB	✓
Get priority	✓
Set priority	✓
Get thread ID	✓
Thread state change handler	✓
Get current stack pointer	✓
Set thread CPU usage	-
Set scheduling mechanism	✓
Lock thread in memory	NA[1]

[1] This feature is irrelevant in VxWorks 5.3.1, since there is never any swapping to disk.

Thread management	YES
Disable scheduling	✓
Total	**15**
Total in percentage	**94%**

3.3.3 Clock and Timer

Clock	YES
Get time of day	✓
Set time of day	-
Get resolution	✓
Set resolution	✓
Adjust time	-
Read counter register	✓
Automatically adjust time	-
Total	**4**
Total in percentage	**57%**

Interval timer	YES
Timer expires on an absolute date	✓
Timer expires on a relative date	✓
Timer expires cyclical	✓
Get remaining time	✓
Get number of overruns	✓
Connect user routine	✓
Total	**6**
Total in percentage	**100%**

3.3.4 Memory Management

Fixed block size partition	YES
Set partition size	✓
Get partition size	✓
Set memory block size	✓
Get memory block size	✓
Specify partition location	-
Get memory block - blocking	-
Get memory block - non blocking	✓
Get memory block - with timeout	-
Release memory block	✓
Extend partition	✓
Get number of free memory blocks	✓
Lock/unlock partition in memory	NA[2]
Total	8
Total in percentage	73%

Non-fixed block size pool	YES
Set pool size	✓
Get pool size	✓
Make new pool	✓
Get memory block size	✓
Get memory block – blocking	-
Get memory block - non blocking	✓
Get memory block - with timeout	-
Release memory block	✓
Extend pool	✓
Extend block	✓

[2] This feature is irrelevant in VxWorks 5.3.1, since there is never any swapping to disk.

Non-fixed block size pool	YES
Get remaining free bytes	✓
Lock/unlock pool in memory	NA[2]
Lock/unlock block in memory	NA[2]
Total	9
Total in percentage	82%

3.3.5 Interrupt Handling

Interrupt handling	YES
Attach interrupt handler	✓
Detach interrupt handler	-
Wait for interrupt - blocking	-
Wait for interrupt - with timeout	-
Raise interrupt	-
Disable/Enable hardware interrupts	✓
Mask/Unmask a hardware interrupt	✓
Interrupt sharing	-
Total	3
Total in percentage	38%

3.3.6 Synchronization and Exclusion Objects

Counting semaphore	YES
Get maximum count	-
Set maximum count	-
Set initial value	✓
Share between processes	✓
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓

Counting semaphore	YES
Post	✓
Post – Broadcast	-
Get status (value)	✓
Total	7
Total in percentage	70%

Binary semaphore	YES
Set initial value	✓
Share between processes	✓
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Post	✓
Get status	✓
Total	7
Total in percentage	100 %

Mutex	YES
Set initial value	-
Share between processes	✓
Priority inversion avoidance mechanism	✓
Recursive getting	✓
Thread deletion safety	✓
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Release	✓
Get status	✓
Get owner's thread ID	✓

Mutex	YES
Get blocked thread ID	✓
Total	**11**
Total in percentage	**92%**

Conditional variable	NO
Pend non blocking	-
Pend with timeout	-
Pend in fifo / priority order	-
Broadcast	-
Priority inversion	-
Simple inquiry	-
Complex inquiry	-
Total	**0**
Total in percentage	**0 %**

Event flags	NO
Set one at a time	-
Set multiple	-
Pend on one	-
Pend on multiple	-
Pend with OR conditions	-
Pend with AND conditions	-
Pend with AND and OR conditions	-
Pend with timeout	-
Total	**0**
Total in percentage	**0 %**

POSIX signals	YES
Install signal handler	✓
Detach signal handler	✓

POSIX signals	YES
Mask/unmask signals	✓
Identify sender	✓
Set destination ID	✓
Set signal ID	✓
Get signal ID	✓
Signal thread	✓
Queued signals	✓
Total	**9**
Total in percentage	**100 %**

3.3.7 Communication and Message Passing Objects

Queue	YES
Set maximum size of message	✓
Get maximum size of message	✓
Set size of queue	✓
Get size of queue	✓
Get number of messages in queue	✓
Share between processes	✓
Receive – blocking	✓
Receive – non blocking	✓
Receive – with timeout	✓
Send - with ACK	-
Send - with priority	✓[3]
Send – OOB (out of band)	✓
Send - with timeout	✓
Send – broadcast	-
Timestamp	-

[3] Message priorities only available in POSIX 1003.1b message queues.

Queue	YES
Notify	✓
Total	**13**
Total in percentage	**81%**

Mailbox	NO
Set maximum message size	**-**
Get maximum message size	**-**
Share between processes	**-**
Send - with ACK	**-**
Send - with timeout	**-**
Send – broadcast	**-**
Receive – blocking	**-**
Receive – non blocking	**-**
Receive – with timeout	**-**
Get status	**-**
Total	**0**
Total in percentage	**0**

3.4 Internet Support

Internet support	0	8	10

The "Tornado for embedded internet" product suite has the components needed for most embedded Internet applications.

Wind River Systems has a product suite called "Tornado for embedded internet". This product suite includes the following components:

- HTTP server for posting information to the Internet/intranet (Wind web server).
- Web browser for viewing information on the World Wide Web/intranet.
- Java support for developing "write once, run anywhere" applications.
- Graphic components for displaying information to the Internet/ intranet user.
- Internet-ready, real-time operating system and development tools.
- Drivers for specialized hardware.
- Networking protocol support for communicating across the Internet/intranet.

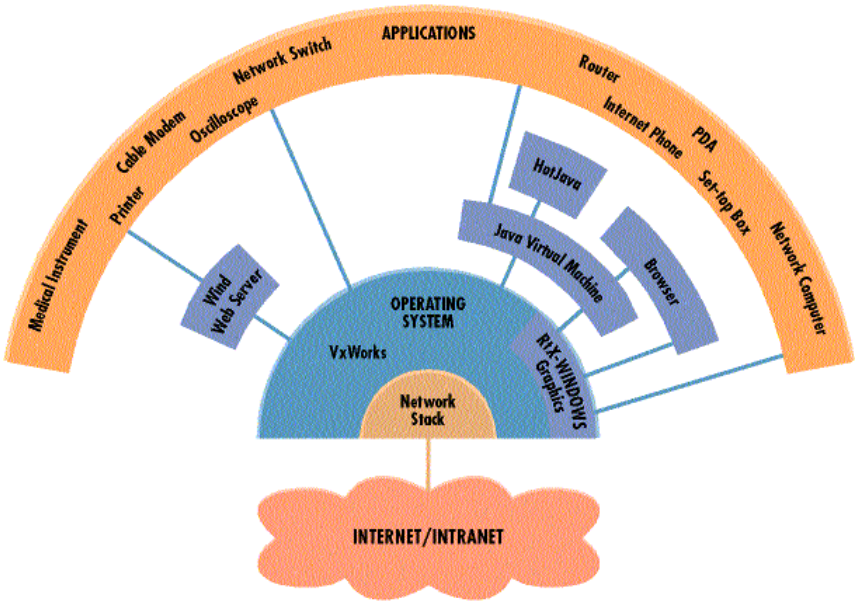

Figure 3.4-1: Tornado for embedded internet suite

In addition, the vendor has informed us that VxWorks has C++ library based graphics, HTML based graphics and Sun certified pJava support.

3.5 Tools

Tools 0 9 10

The Tornado development environment comes with a very rich set of tools. Moreover, Tornado is a completely open environment making it easy to integrate development tools from third party vendors.

Wind River has an integrated development environment for embedded applications called “Tornado”. Tornado is a completely open environment designed to be customized and extended by the developer. Its open interfaces make it possible to integrate other development tools from third parties.

Tornado adopted the tool command language (Tcl) for easy extensibility. Much of the Tornado facilities are implemented in Tcl. The source code is included because Tcl is an interpreted language. This allows customers to add new commands and facilities, or to change the way existing features work to better fit their needs.

Along with the tools checked in the table below, Tornado provides additional tools to perform real-time data analysis (Stethescope), tools to detect memory leaks and code coverage tools.

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Editor					
Color highlight	**Yes**	✓			✓
Integrated help[4]	**Yes**	✓			✓
Automatic code layout	**No**				
Compiler					
C	**Yes**	✓	✓	✓	✓
C++	**Yes**	✓	✓	✓	✓
Ada	**No**				
Assembler	**Yes**		✓	✓	
Linker					
Incremental	**Yes**	✓	✓	✓	✓
Symbol table generation	**Yes**	✓	✓	✓	✓

[4] No online help for system calls is available, only for issues pertaining to general usage of Tornado.

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Development tools					
Profiler	**Yes**	✓			✓
Project management	**Yes**	✓			✓
Source code control	**No**[5]				
Revision control	**No**[5]				
Debugger					
Symbolic debugger	**Yes**	✓			✓
Thread sensitive debugging	**Yes**	✓			✓
Mixed source and disassembly	**Yes**	✓			✓
Variable inspect	**Yes**	✓			✓
Structure inspect	**Yes**	✓			✓
Memory inspect	**Yes**	✓			✓
Register inspect	**Yes**	✓			✓
System analysis tool					
Tracing	**Yes**	✓			✓
Thread information	**Yes**	✓			✓
Interrupt information	**Yes**	✓			✓
Loader					
TFTP boot loader	**Yes**	✓			✓
BOOTP loader	**Yes**	✓			✓
Serial boot loader	**Yes**	✓			✓
Load separate modules	**Yes**	✓			✓

[5] The vendor has informed us that this feature will be available in Tornado II

3.6 Documentation and Support

Documentation and Support	0	7	10

The manuals are extensive but not always clear. Users can get access to a huge knowledge base on the Wind River website. Technical support is very professional.

Tornado 1.0.1 comes with an extensive set of manuals. Unfortunately, the manuals are not always clear and comprehensive. For example: the reference manual for the API and library calls does not always provide an explanation of the function parameters. Very little information is available on the architecture of the system. The reader may find himself browsing through various manuals before finding the information he was looking for.

☺ The Wind River web site (http://www.wrs.com) has a large database of online publications, FAQs and more. Customers with a valid maintenance contract can get a username and password that provides access to this database.

☺ Customer support is professional. Support is best requested via email. Tornado support request (TSR) messages can be sent from within the Tornado environment. Before sending the email to Wind River technical support, the TSR tool adds specific information about your host and target system to your message that might be helpful to the technical support representative.

3.7 Development methodology

Development methodology	0	8	10

Complete development environment available for both Windows and UNIX based hosts. Tools like dynamic linking and loading shorten the test and debug cycle. The availability of VxWorks simulators allows development of applications to begin without the hardware being available.

WindRiver Systems uses the *host ≠ target* approach. Host and target are two different machines linked together (serial, LAN, bus, etc) for communication. The host is the machine on which the development environment (Tornado 1.0.1) runs. Tornado is available for both Windows (including Windows 3.1, Windows 95, Windows 98 and Windows NT) and UNIX based hosts. The target is the machine which runs the dedicated RTOS (VxWorks 5.3.1) with the real-time application.

The advantage of this approach over the host = target approach is that a separate host is more suitable as a system development host. Indeed, all the features of the GPOS (Windows NT or UNIX) can be used, which allows for a better and more complete development environment.

In this configuration however, the debugger is on the host, while the real-time application is executed on the target. To make interactive debugging possible, VxWorks installs so called “debug agents” on the target that communicate debug information to the host. These debug agents provide a virtual seamless integration of host and target while debugging i.e., the user is under the impression that he is debugging a local application.

A nice feature of VxWorks/Tornado is dynamic linking and loading of modules. This feature can reduce the edit-test-debug cycle. The user can download an individual object module into the target without having to reboot. The module is dynamically linked into the target. There is no need for the user to compile and link the complete executable on the host and download it to the target.

To assist developers of embedded systems using custom hardware, WindRiver Systems also offers VxSim, which is a prototyping and simulation tool for Tornado/VxWorks. VxSim provides a simulation of VxWorks on the host. With this tool, application development can begin before the hardware becomes available. VxSim was not tested for this evaluation.

4 Practical evaluation

Test Results	0	7	10

Predictable behavior. The semaphore queue management could be improved : the time it takes to add a thread to the queue (and sort it), is proportional with the number of threads in the queue. Clock ISR execution time can get pretty lengthy when timers are used, slowing down the system.

In order to clarify the content of the following sections, the reader is referred to the document "report definition and test plan", which can be downloaded from the visitor center (http://www.dedicated-systems.com/encyc). This document defines (among other things) the different memory protection models, the objects and features mentioned in our API richness section. It also contains a detailed explanation of the tests performed.

4.1 System under test

This section describes how the operating system and test equipment was configured during the tests.

4.1.1 Operating system configuration

The boot parameters were configured as follows:

- Boot device : ene
- Unit number : 0
- Processor number : 0
- Flags : 0x80 (use tftp to get boot image)

4.1.2 Test configuration

All the tests were executed using the following hardware:

- Motherboard: Chaintech 5TTMT M201 with a 33MHz PCI bus
- BIOS: Award BIOS v4.51PG
- CPU: Intel Pentium 200Mhz MMX Family 5 Model 4 Stepping 3 (with 32KB L1 Cache)
- RAM: 32 Mb DIMM
- L2 Memory Cache: 512KB
- Hard drive: Western Digital Caviar 34300, primary master, UDMA 2
- Graphic adapter: S3 Trio 64V2/DX in PCI slot 1
- Network interface card: SVEC NE2000 Plug & Play ISA network card
- PCI exerciser in PCI slot 3 (PCI interrupt level D, local bus interrupt level 10)
- PCI analyzer in PCI slot 4
- External and CPU internal cache was enabled during the tests, unless otherwise specified. All tests were performed while connected to a 10 Mbit/s ethernet local area network.

http://www.dedicated-systems.com
email: info@dedicated-systems.com

– DRAM timing was set to 60ns.

4.1.3 Additional comments

According to the document “Report definition”, which explains the details on the tests performed and is available from our website, some tests are supposed to be executed using different memory protection models, to demonstrate the overhead caused by memory protection.

This is not the case in this paper, since only memory protection model ‘a’ (no memory protection between system and application) is available in VxWorks 5.3.1 by default. An optional component (VxVMI) is necessary for memory protection.

4.2 Interrupt handling – Without rescheduling

Due to our measuring method for interrupt handling, only a few hundred samples for both interrupt latency and dispatch latency were obtained. This as opposed to several thousand samples for the other tests. The reason for this is explained in the document "Report definition and test plan", section 5.2.2.

For this test we created a high priority thread that continuously writes traces to the PCI bus (to be captured and time stamped by our PCI analyzer). Interrupts are being generated roughly every 50 µs, interrupting the thread. Every time the interrupt occurs, the interrupt handler starts executing, writes its trace to the PCI bus, and returns. After the interrupt handler has finished executing, the thread that was interrupted resumes again. The system doesn't reschedule. This test measures the interrupt latencies while the system is heavily loaded, since the main (high priority) thread is constantly writing traces to the bus claiming all available CPU power.

The metrics measured are the *interrupt latency* and the *interrupt dispatch latency*. The *interrupt latency* is the time it takes to go from an interrupted thread to the interrupt handler, while the *interrupt dispatch latency* is the time needed to get back from the interrupt handler to the thread scheduled for execution.

Interrupt handling in VxWorks 5.3.1 is fast. There are a few reasons for this. First of all, there is no MMU memory protection (unless an optional component VxVMI was installed) which reduces the overhead and results in faster context switches (see 3.2.2.2). Second, the interrupt handlers run in a context outside of any thread's context (see 3.2.2.3). So no thread context switches are involved when an interrupt occurs.

A maximum of only 2.7 µs was measured for the interrupt latency, while the interrupt dispatch latency peaked at 2.5 µs. Both latencies averaged around 1.9 µs (on a 200 MHz Pentium MMX).

The reader should bear in mind that on a PC-based target (like the one that was used for these tests), the clock interrupt is assigned the highest priority (higher than all other external interrupts). In such architectures, the time needed to service this clock interrupt should be added to the interrupt latencies to get the worst case scenario results. We have noticed that on the average it takes about 4 µs for VxWorks to service the clock interrupt (on a Pentium 200 MHz), but one incident where it took longer than 20 µs was observed (see section 4.9.3). We did not generate an interrupt simultaneously with the clock interrupt, so this is not demonstrated in this test.

For a detailed discussion on the execution time of the clock ISR and how this may impact the performance, the reader is referred to section 0.

4.2.1 Interrupt latency (IL-a-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
157	1860	2760	1889.8

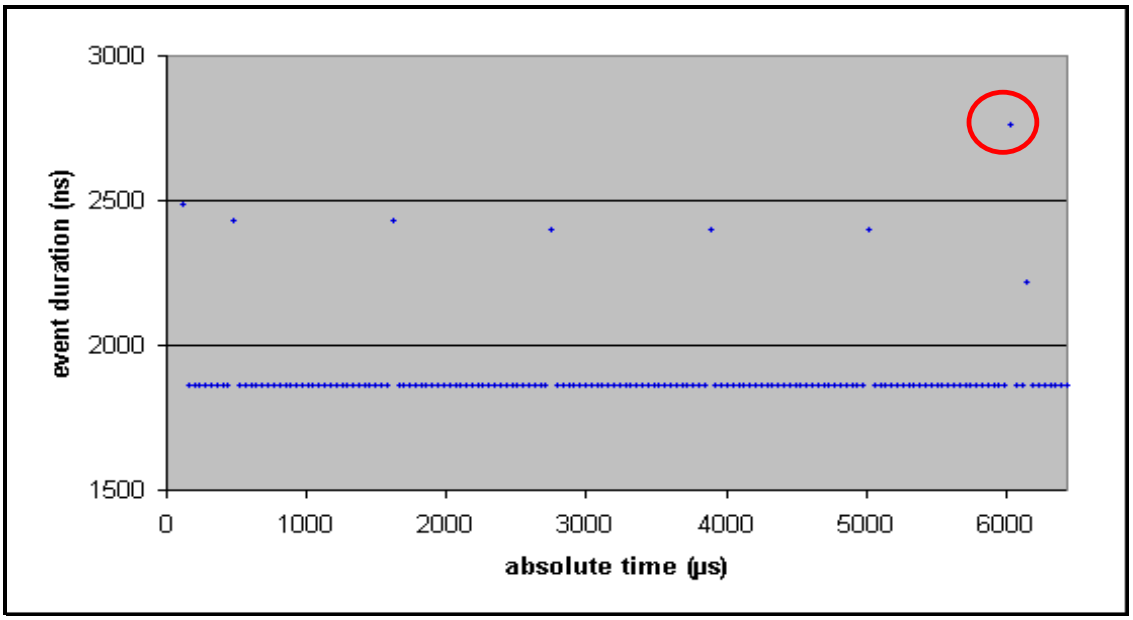

Figure 4.2-1 IL-a-1.a

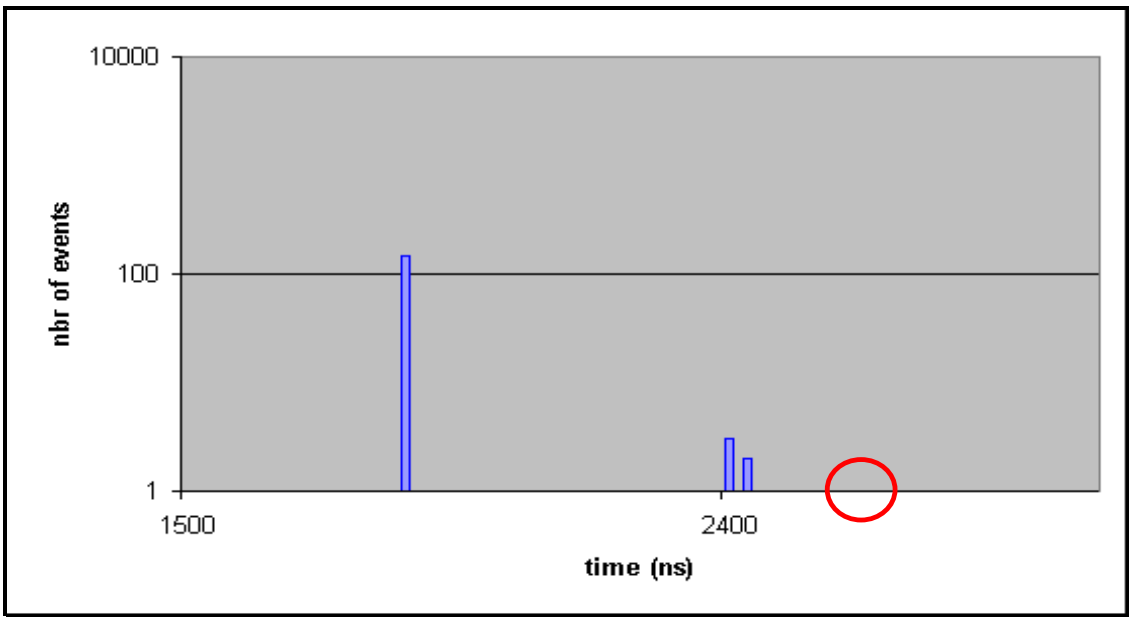

Figure 4.2-2 IL-a-1.a - frequency distribution

4.2.2 Interrupt dispatch latency (IDL-a-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
157	1830	2460	1897.1

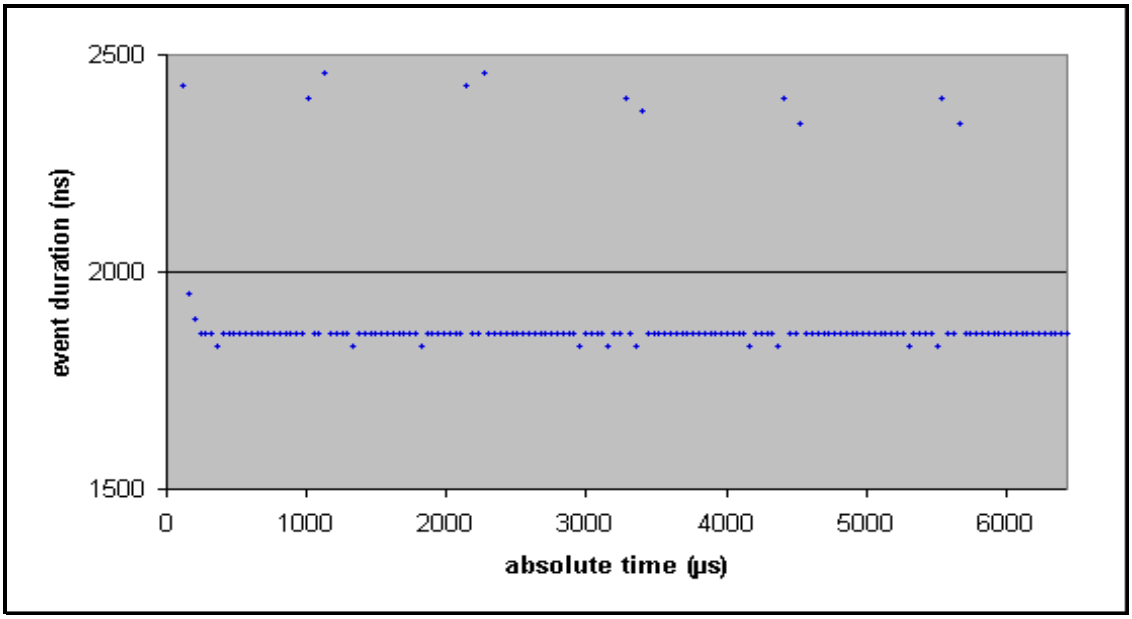

Figure 4.2-3 IDL-a-1.a

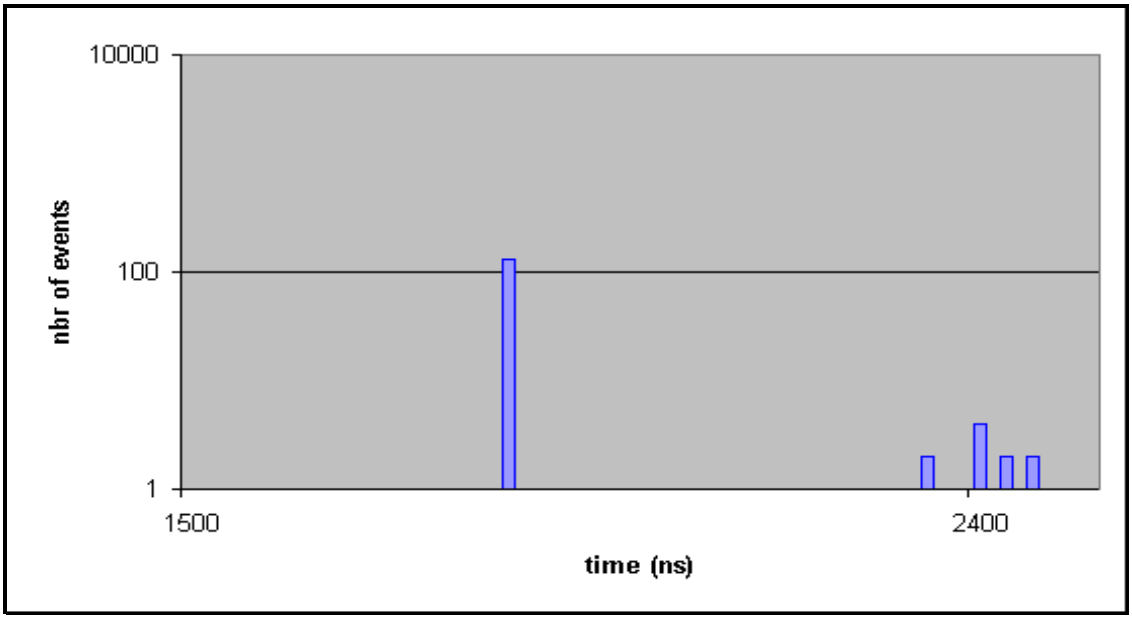

Figure 4.2-4 IDL-a-1.a - frequency distribution

http://www.dedicated-systems.com
email: info@dedicated-systems.com

4.3 Interrupt handling – Rescheduling

In this test series, a low priority thread is interrupted. As soon as the interrupt handler is activated, it releases a semaphore on which a higher priority thread was waiting. When the ISR returns, the system reschedules to let the high priority thread run. The metric that is measured is the interrupt dispatch latency, i.e. the time needed to go from the interrupt handler to the next scheduled thread. The test was executed with 1, 10 and 128 threads of equal priority pending on the semaphore.

The interrupt dispatch latency in this test is bigger than in the previous test because of the rescheduling taking place. The peaks are due to the clock interrupting the measurement. The clock interrupt has the highest priority level in PC-based targets.

The average dispatch latency seems to rise a little bit as the number of threads is increased. This is due to caching: when using only a limited number of threads, every thread's context fits in cache memory. This is not the case when the number of threads is increased significantly. This is demonstrated in section 4.5, where thread switch latencies are handled and tests are executed with cache memory enabled and disabled.

4.3.1 Interrupt dispatch latency (IDL-b-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	2700	7140	2755.8

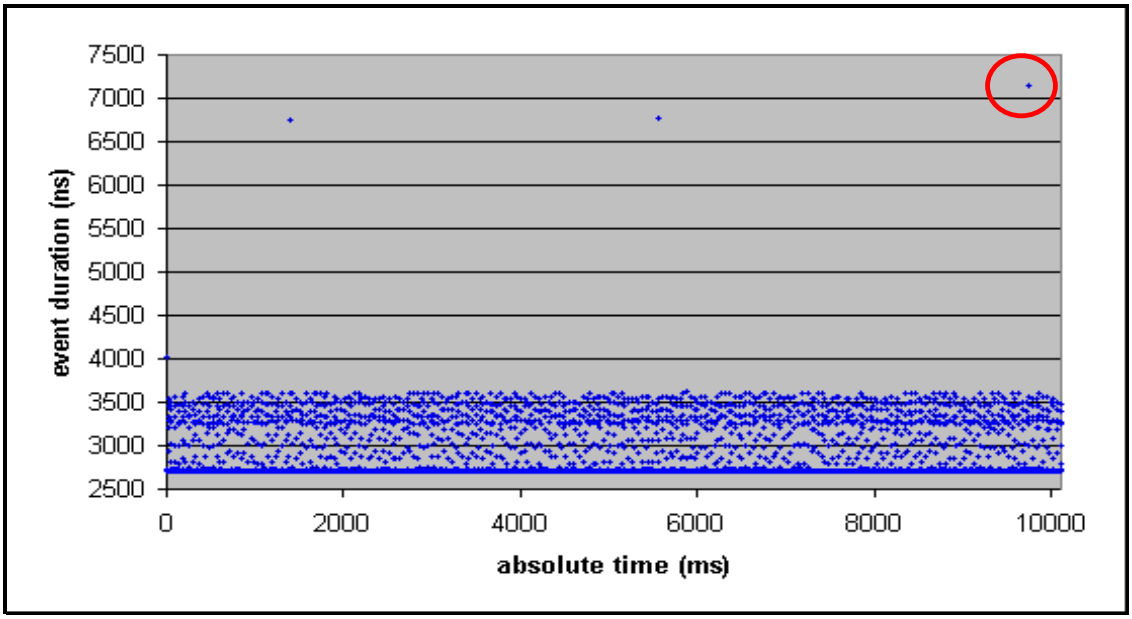

Figure 4.3-1 IDL-b-1.a

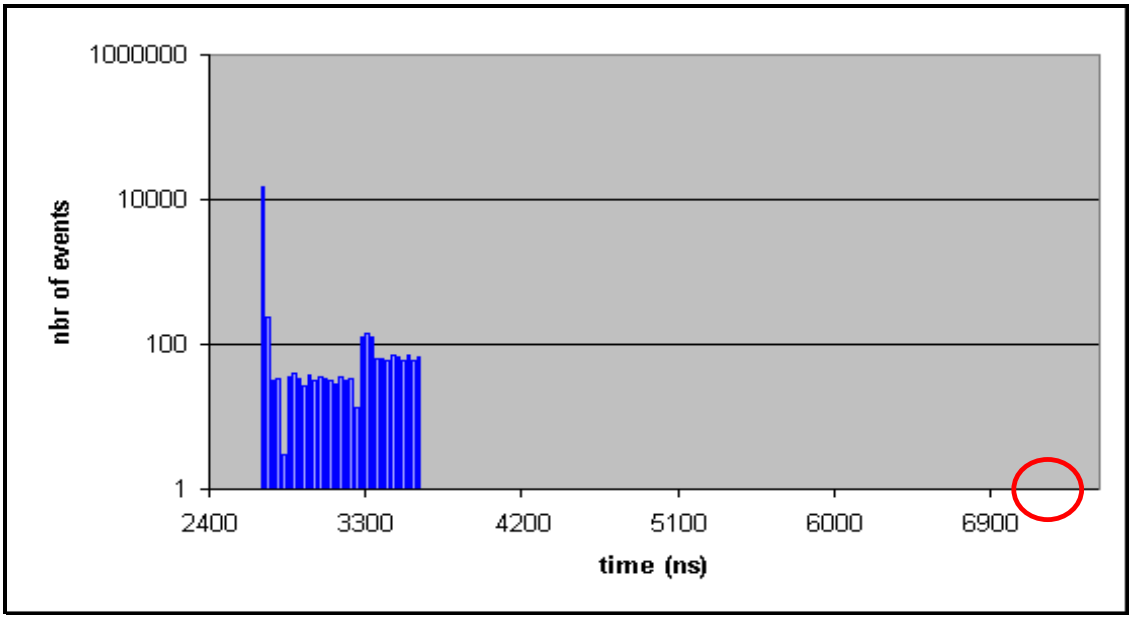

Figure 4.3-2 IDL-b-1.a - frequency distribution

4.3.2 Interrupt dispatch latency (IDL-b-10.a)

- Number of threads: 10
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	2700	13080	2783.5

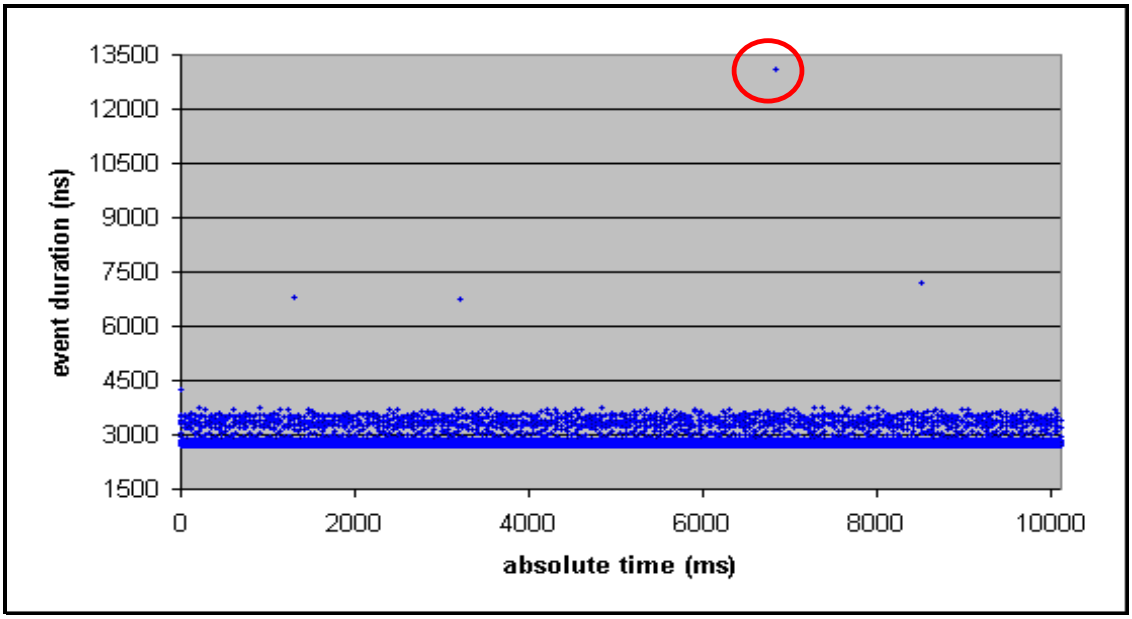

Figure 4.3-3 IDL-b-10.a

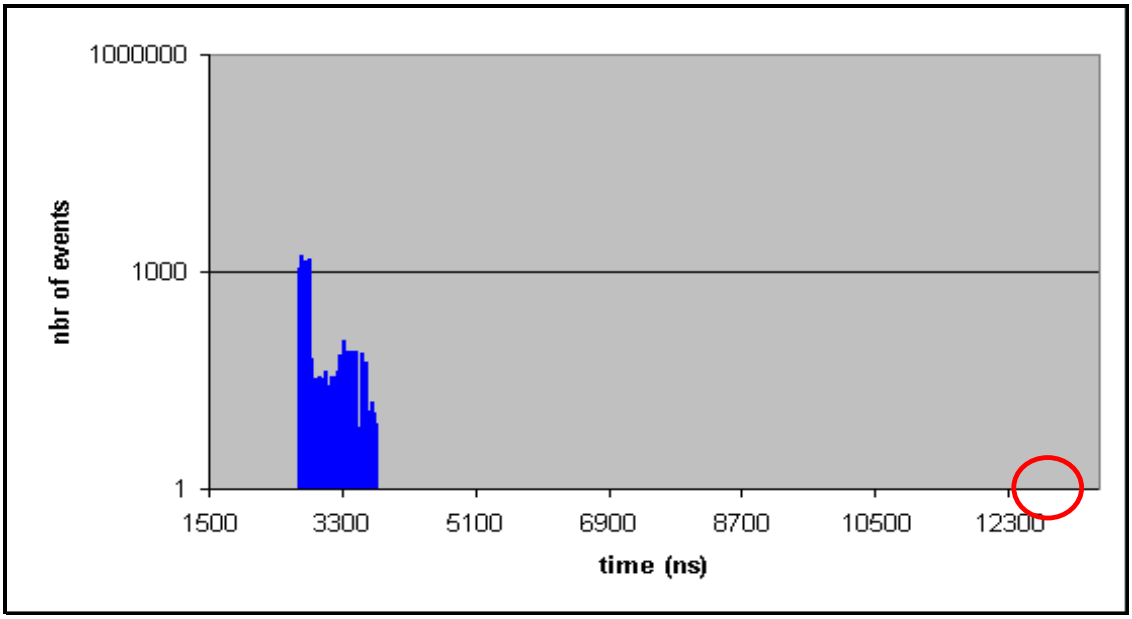

Figure 4.3-4 IDL-b-10.a - frequency distribution

4.3.3 Interrupt dispatch latency (IDL-b-128.a)

- Number of threads: 128
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	2820	8430	3192.3

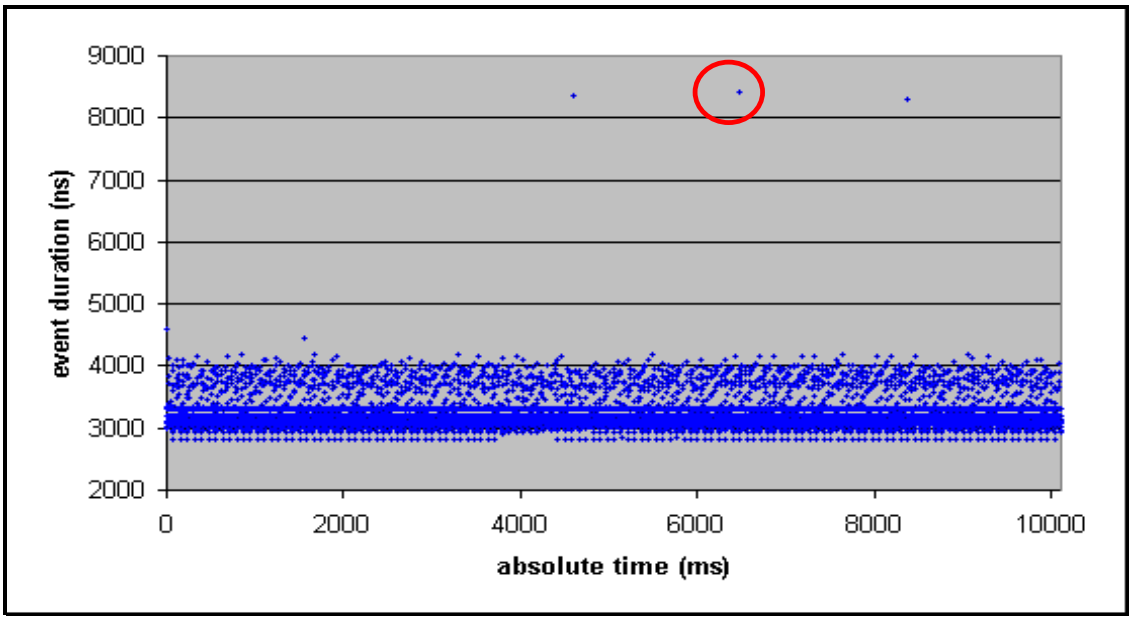

Figure 4.3-5 IDL-b-128.a

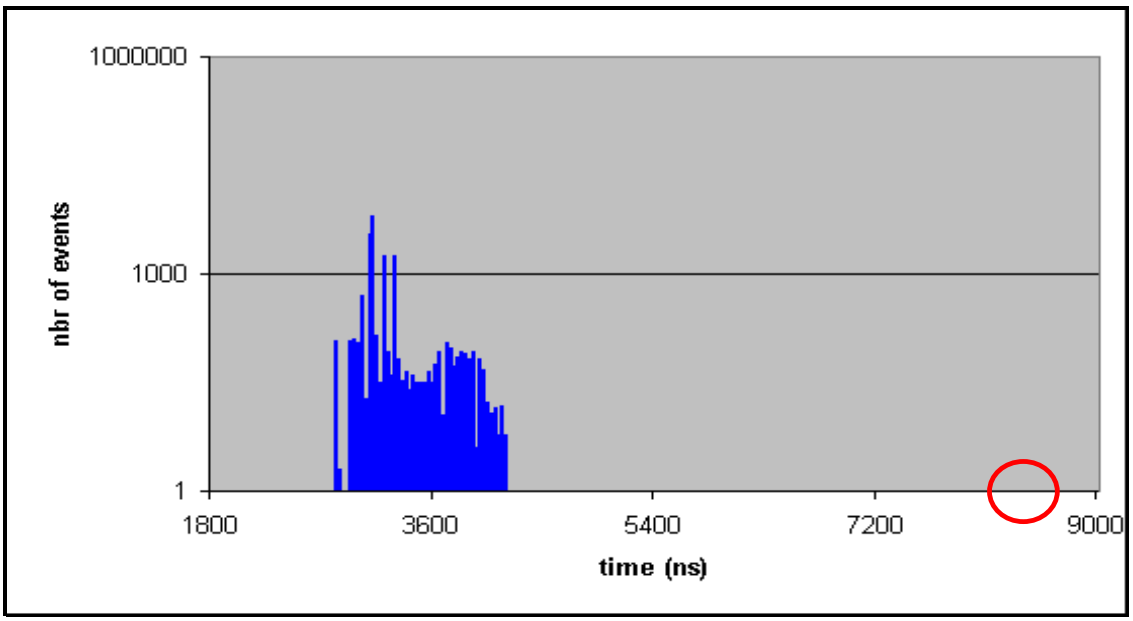

Figure 4.3-6 IDL-b-128.a - frequency distribution

http://www.dedicated-systems.com
email: info@dedicated-systems.com

4.4 Thread

The tests in this section handle thread creation and deletion.

For all the tests in this section it can be observed that the first measurement for the creation or deletion of a thread is much higher than the average. This is mainly due to caching. The maximum is of course of much greater importance than the average in real-time systems.

There is no major difference between the time it takes to delete a thread that has never executed and a thread that did execute. This can be seen in sections 4.4.2 and 4.4.3. This suggests that the thread is created in its entirety when the API call to create the object is executed. Some implementations in other RTOSs only create a minimal object at first, and defer the creation of additional resources that may be necessary until the object is first used. This is not the case in VxWorks 5.3.1.

4.4.1 Creation (TF-a-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8192	30300	60330	30825.0

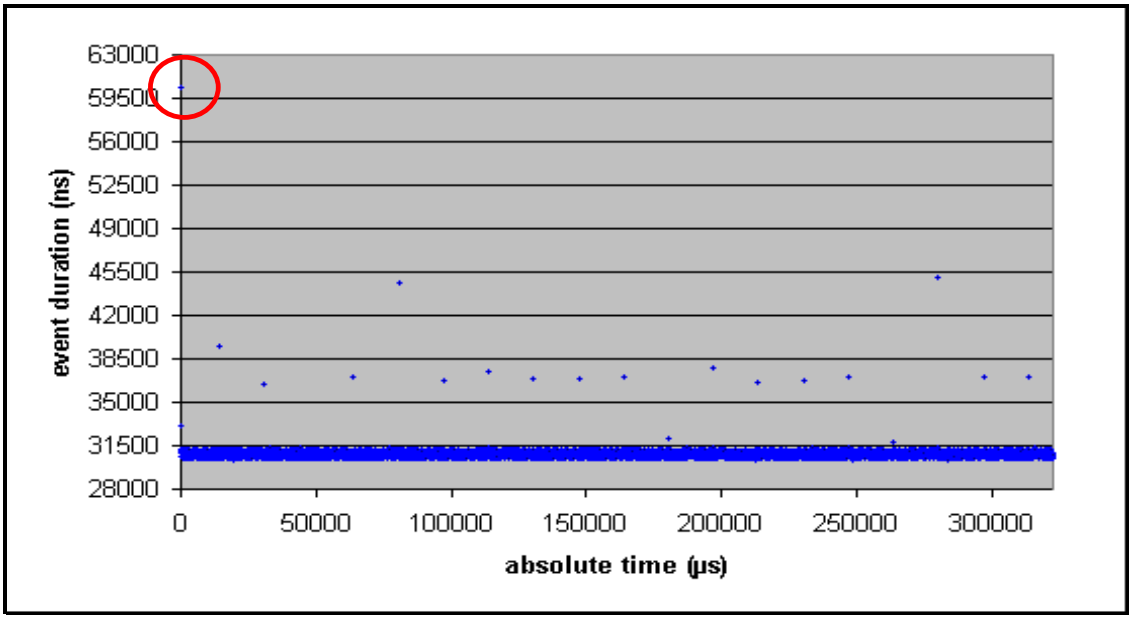

Figure 4.4-1 TF-a-1.a

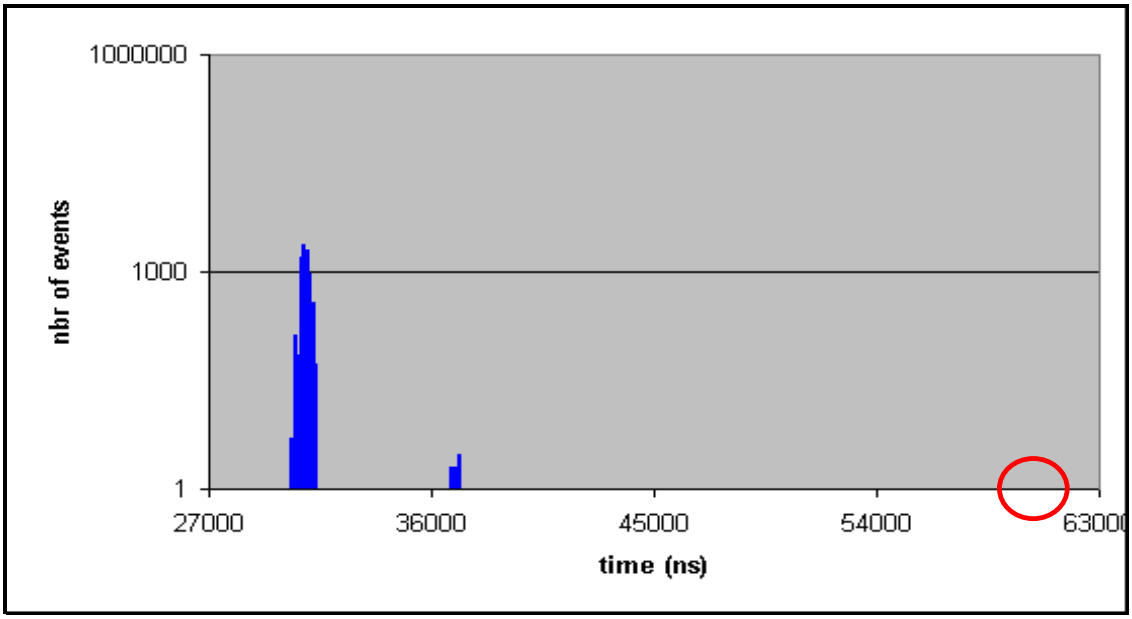

Figure 4.4-2 TF-a-1.a - frequency distribution

4.4.2 Deletion (TF-b-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	7560	19530	8215.5

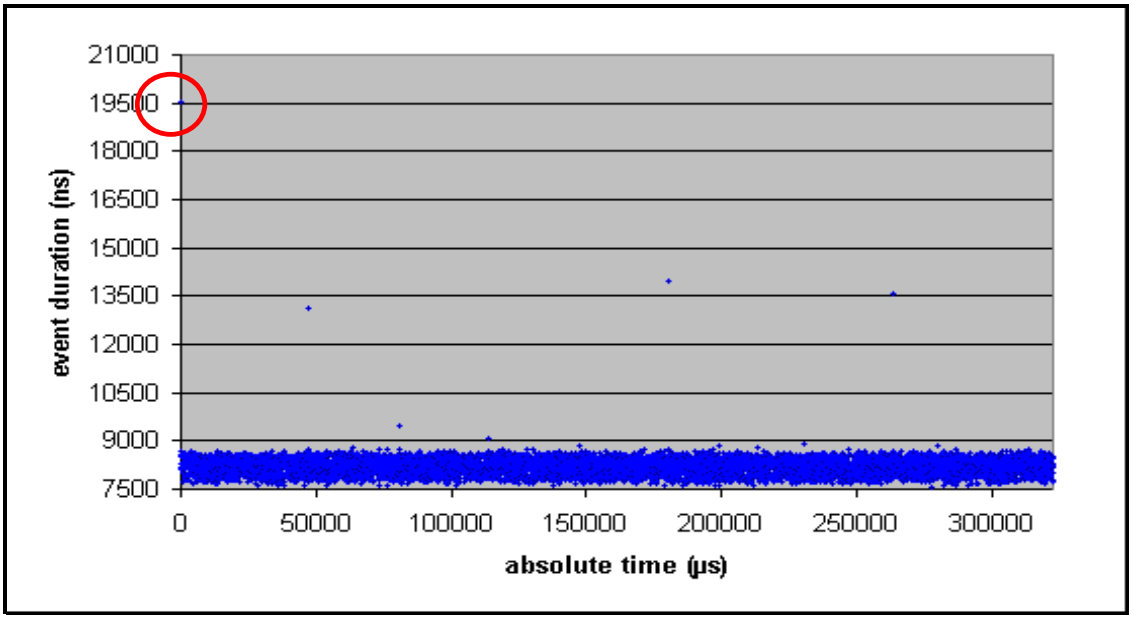

Figure 4.4-3 TF-b-1.a

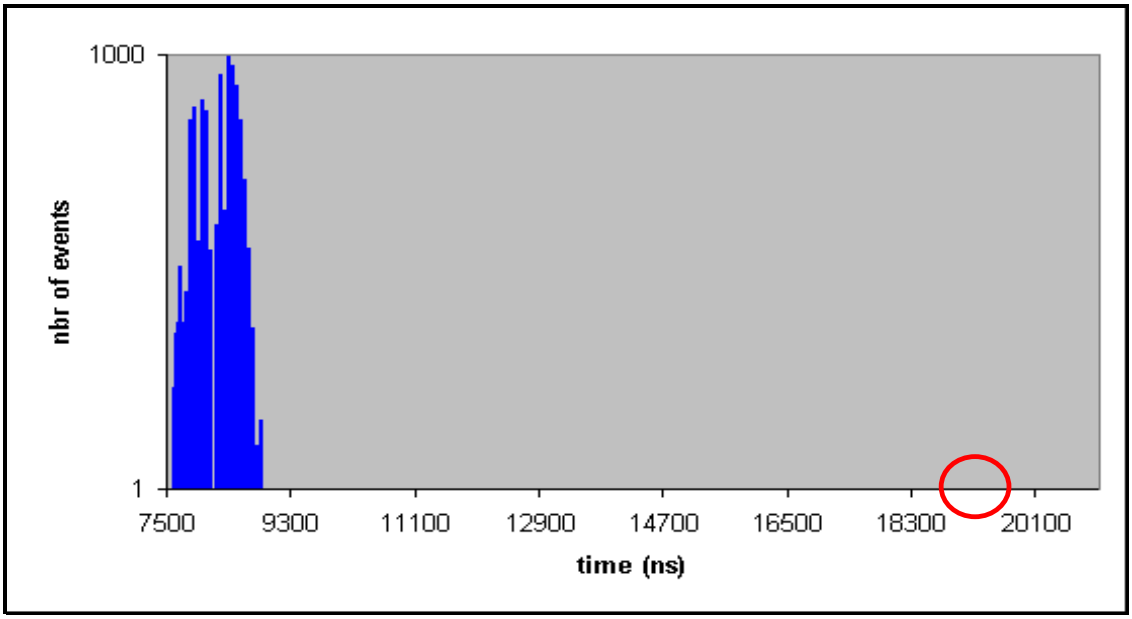

Figure 4.4-4 TF-b-1.a - frequency distribution

4.4.3 Deletion - after use (TF-c-2.a)

- Number of threads: 2
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
6553	8370	16770	9528.9

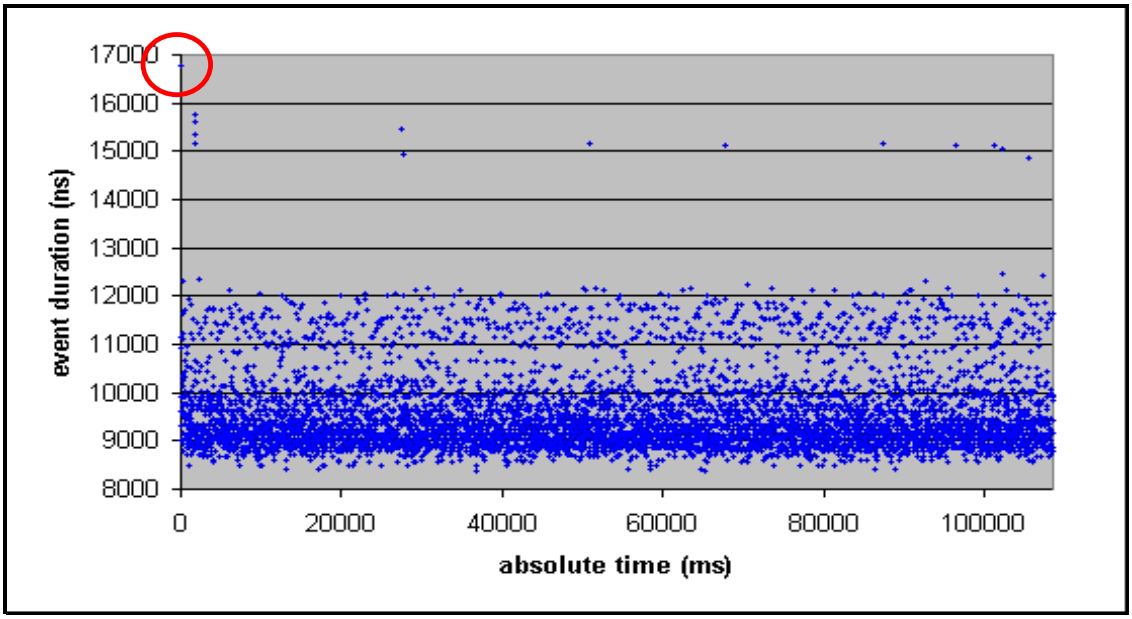

Figure 4.4-5 TF-c-2.a

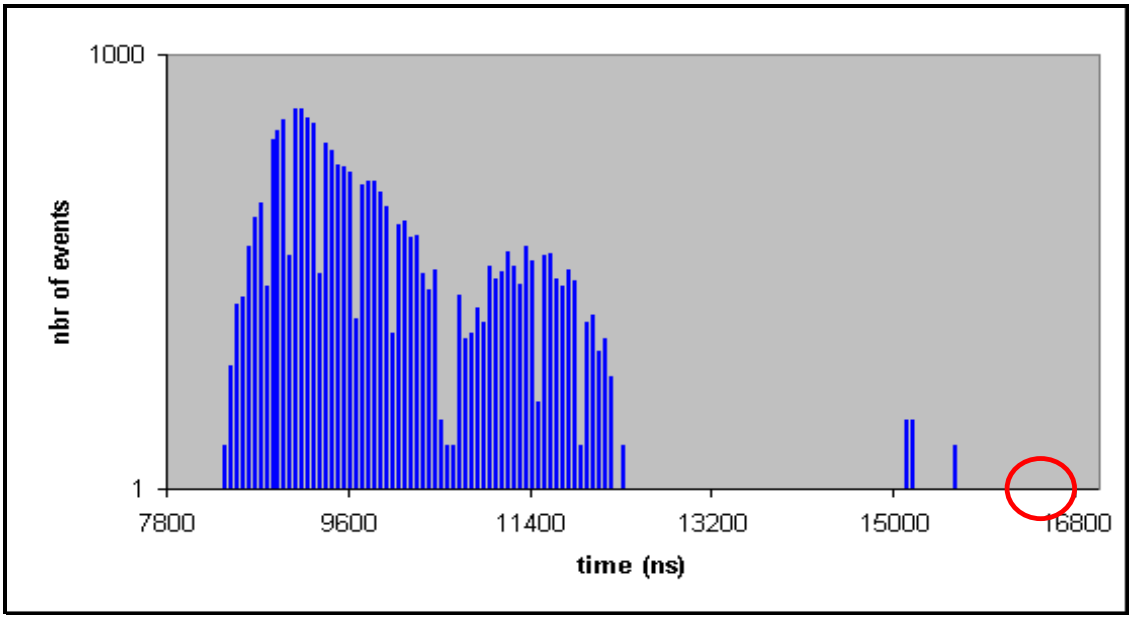

Figure 4.4-6 TF-c-2.a - frequency distribution

4.5 Thread switch latency – same process

In this test series, a number of threads of equal priority are created. As soon as anyone of these threads becomes active, it voluntarily releases the processor to let another thread of the same priority run. This test is executed with 2, 10 and 128 threads. The metric measured is the thread switch latency, i.e. the time it takes to switch from one thread to another.

It is important to know that all the threads run in the same process. This is the default situation in VxWorks 5.3.1 since all the threads share the same memory space by default. An extra component (VxVMI) is required if the user wants to create separate processes that have their own virtual memory.

From the test results in sections 4.5.1, 4.5.2 and 4.5.3 which measure the metric for 2, 10 and 128 threads respectively, it seems that the more threads are created, the longer it takes to perform the thread switch.

However, this is not due to VxWorks' scheduler, but due to caching instead. When the test is executed with a small number of threads, the context from every thread fits in the data cache memory. When the number of threads is increased sufficiently, this is not the case anymore. Therefore, the average thread switch latency will rise.

To proof this, the tests were repeated with both internal and external cache memory disabled. The results are displayed in sections 4.5.4, 4.5.5 and 4.5.6. There, the average thread switch latency remains the same when the number of threads is increased. The cache in VxWorks 5.3.1 should be enabled/disabled by using the appropriate cache library functions. These calls enable/disable both internal and external cache. By default, all cache is enabled, even if it is disabled by the user in the BIOS setup.

The peaks in all the results are due to the clock interrupt being serviced. In a PC architecture (the test platform we used), the clock interrupt has the highest priority, higher than all the other hardware interrupts. This means that in such an architecture, the time it takes to service the clock interrupt should be added to all the latencies to get the worst case scenario. The clock was configured to tick every 16.7 ms. The clock ISR takes an average of about 4 µs to get serviced (on a Pentium 200 MHz MMX) when cache memory is enabled. But in some tests executed for this evaluation, we've seen that it can sometimes take more than 20 µs to get the clock interrupt serviced. An example of this is the test is section 4.9.3 (p. 64) which handles priority inversion for a mutex : while the average time to handle the clock interrupt is only 5 µs, on one occasion it took 22 µs.

For a detailed discussion on the execution time of the clock ISR and how this may impact the performance, the reader is referred to section 0.

4.5.1 Thread switch latency (TSL-a-2.a)

- Number of threads: 2
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	2550	12420	2551.0

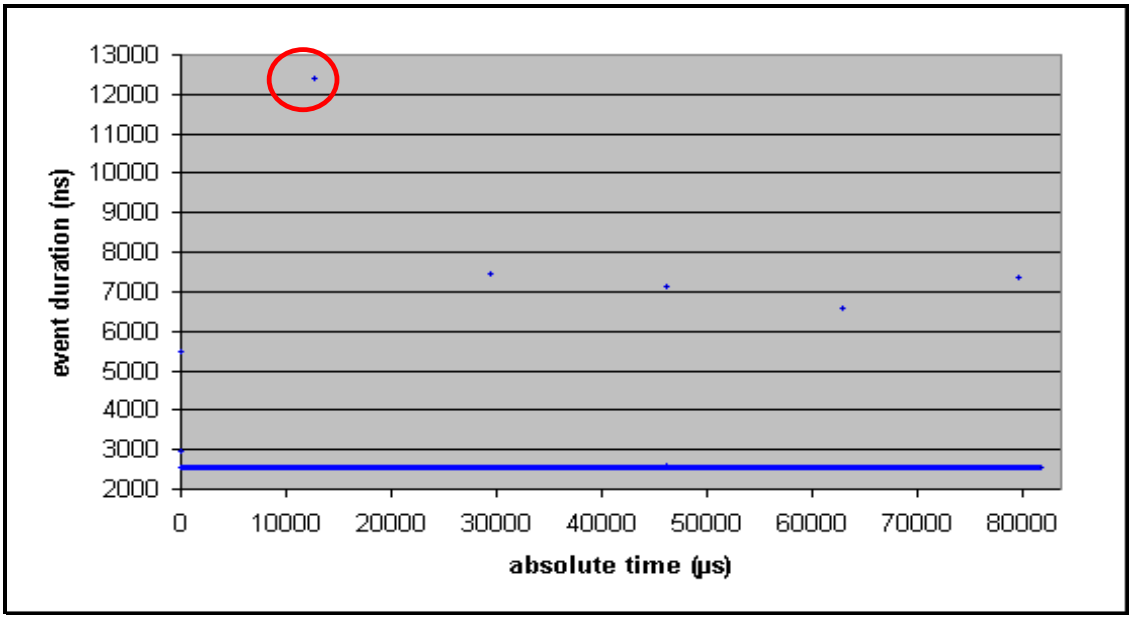

Figure 4.5-1 TSL-a-2.a

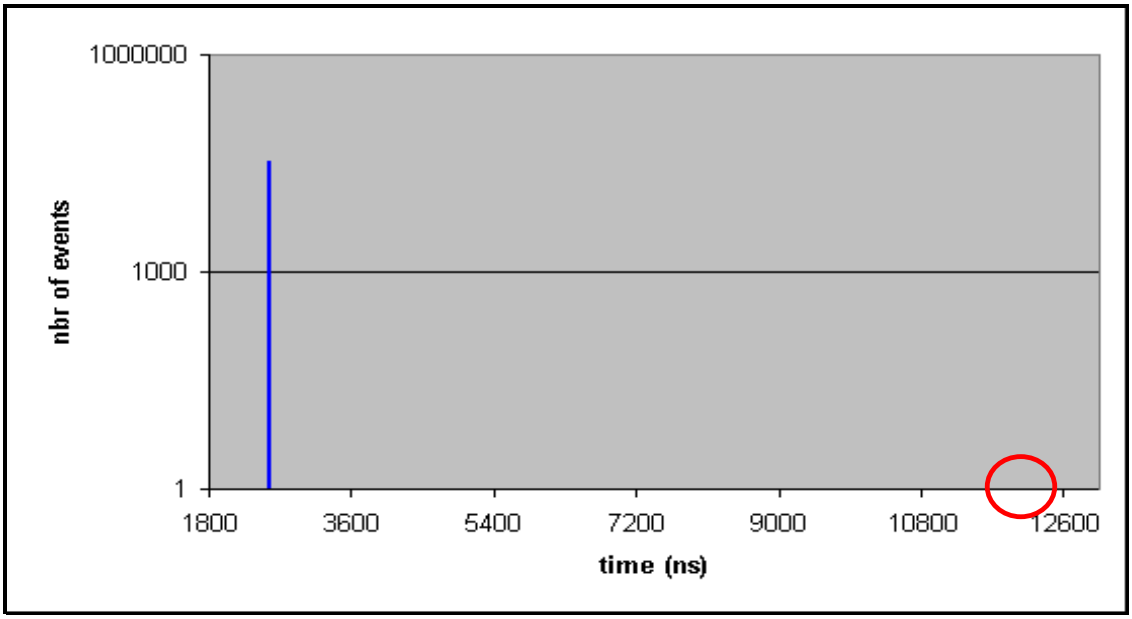

Figure 4.5-2 TSL-a-2.a - frequency distribution

4.5.2 Thread switch latency (TSL-a-10.a)

- Number of threads: 10
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	2460	16020	2551.7

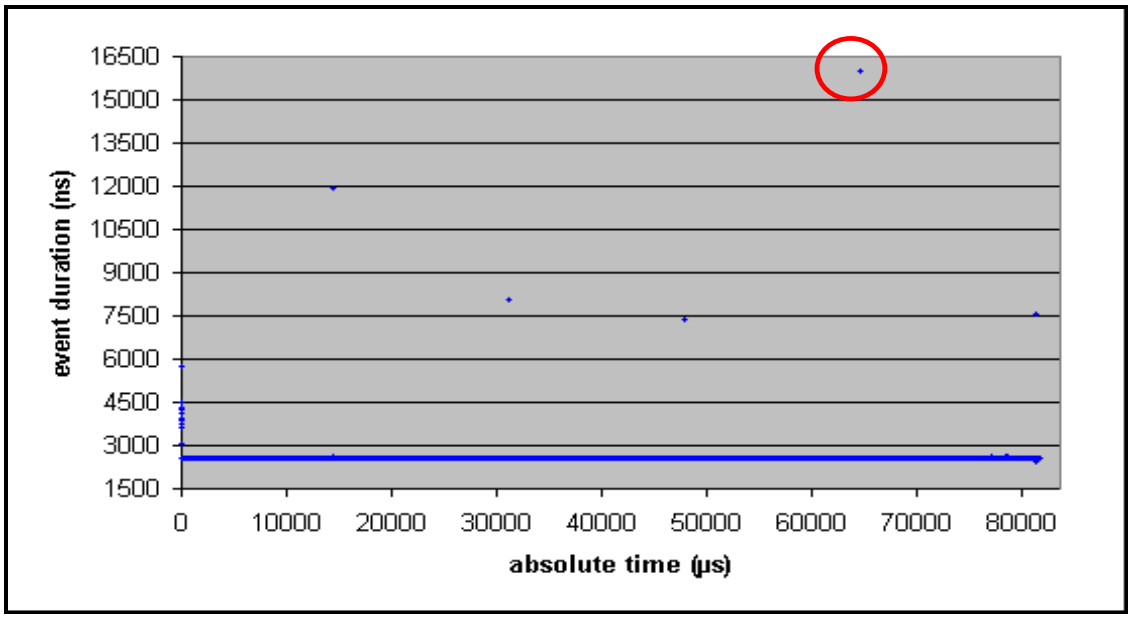

Figure 4.5-3 TSL-a-10.a

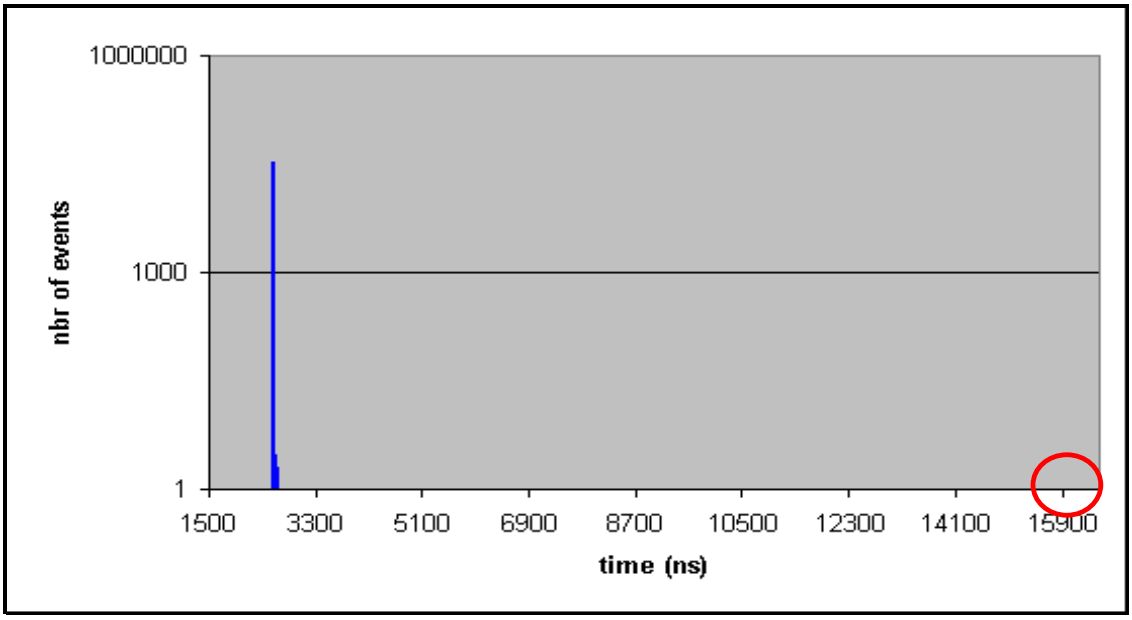

Figure 4.5-4 TSL-a-10.a - frequency distribution

4.5.3 Thread switch latency (TSL-a-128.a)

- Number of threads: 128
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	3120	28380	3328.7

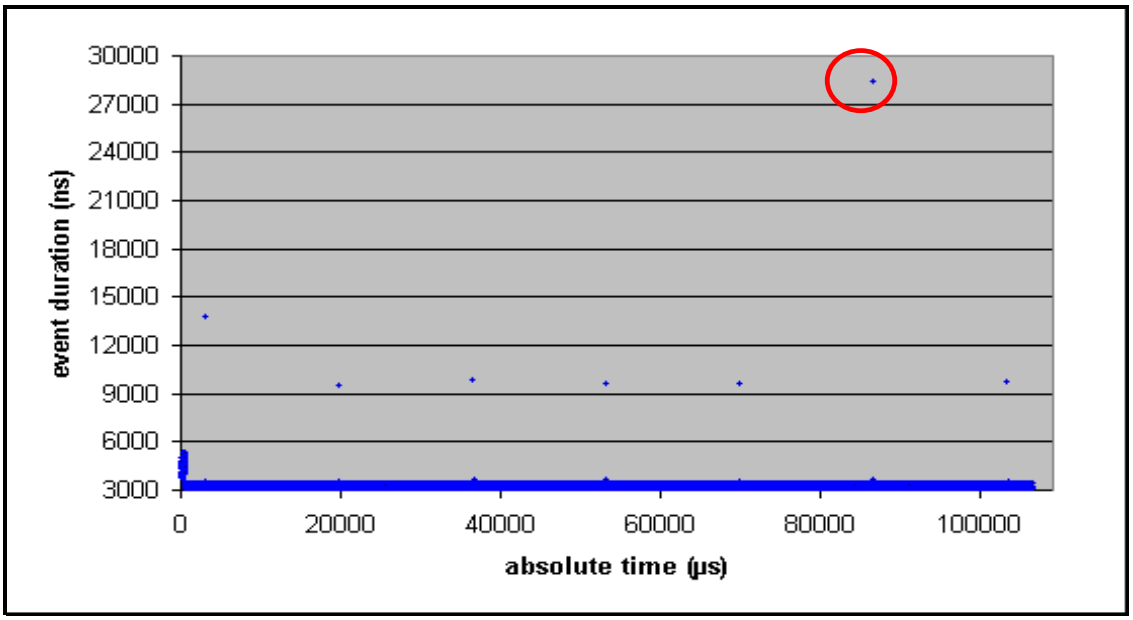

Figure 4.5-5 TSL-a-128.a

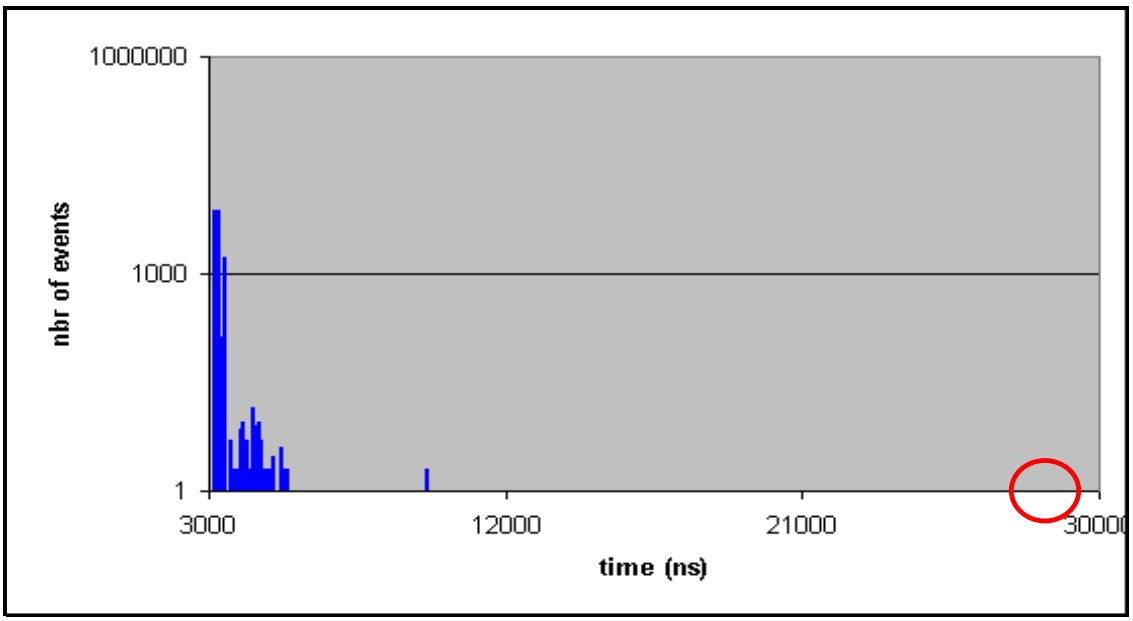

Figure 4.5-6 TSL-a-128.a - frequency distribution

4.5.4 Thread switch latency – cache disabled (TSL-a-2.a)

- Number of threads: 2
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	13020	73710	13067

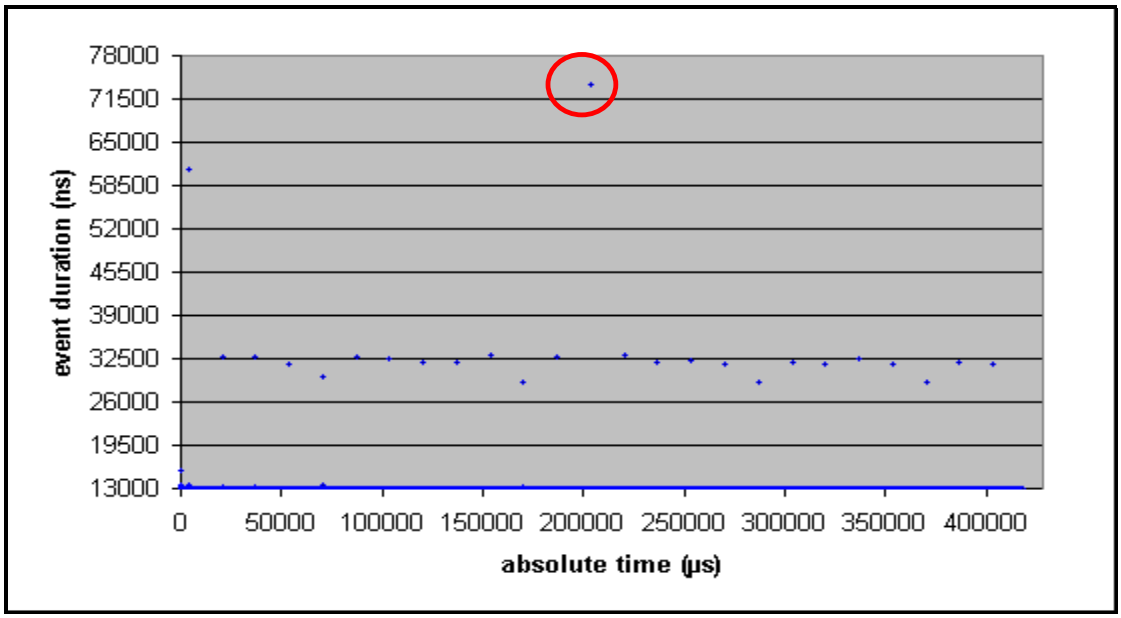

Figure 4.5-7 TSL-a-2.a – cache disabled

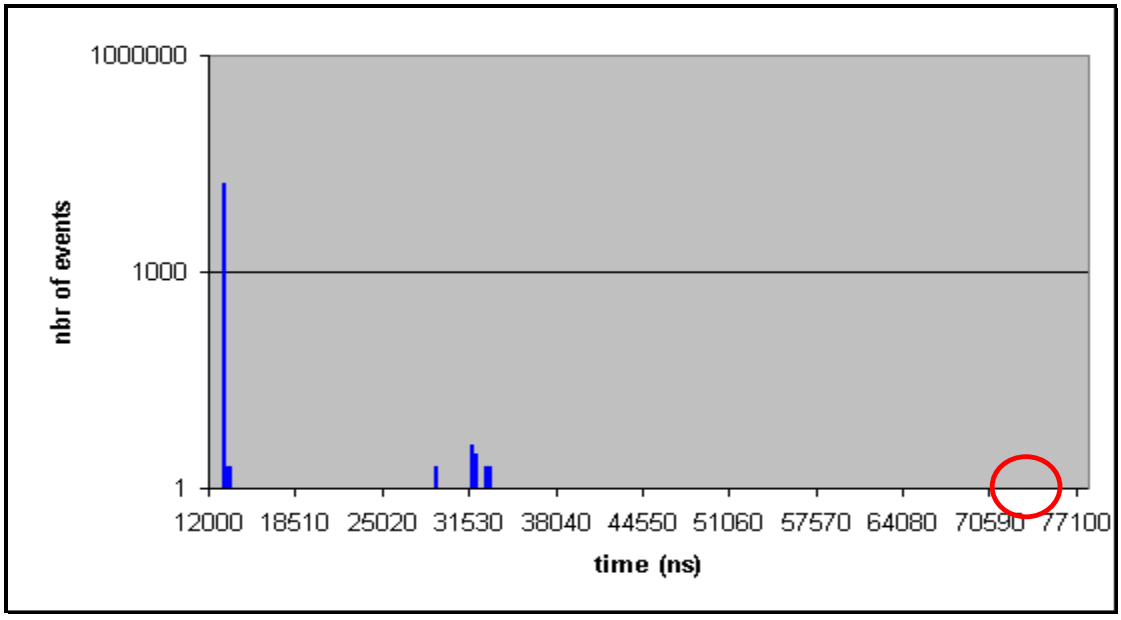

Figure 4.5-8 TSL-a-2.a – cache disabled - frequency distribution

4.5.5 Thread switch latency – cache disabled (TSL-a-10.a)

- Number of threads: 10
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	12960	55530	13006

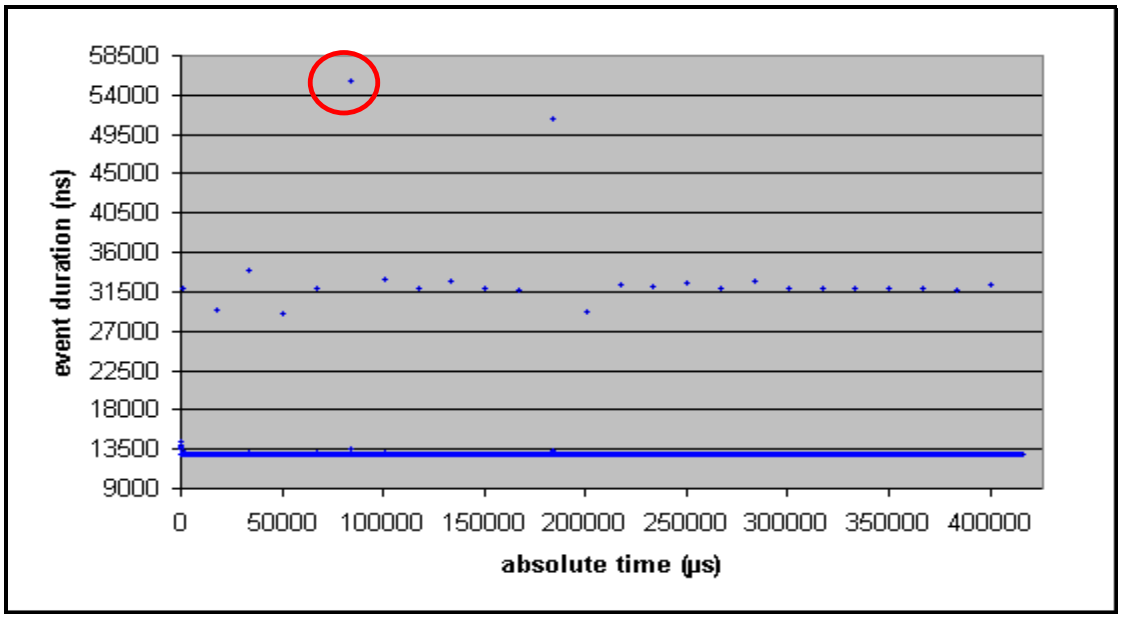

Figure 4.5-9 TSL-a-10.a – cache disabled

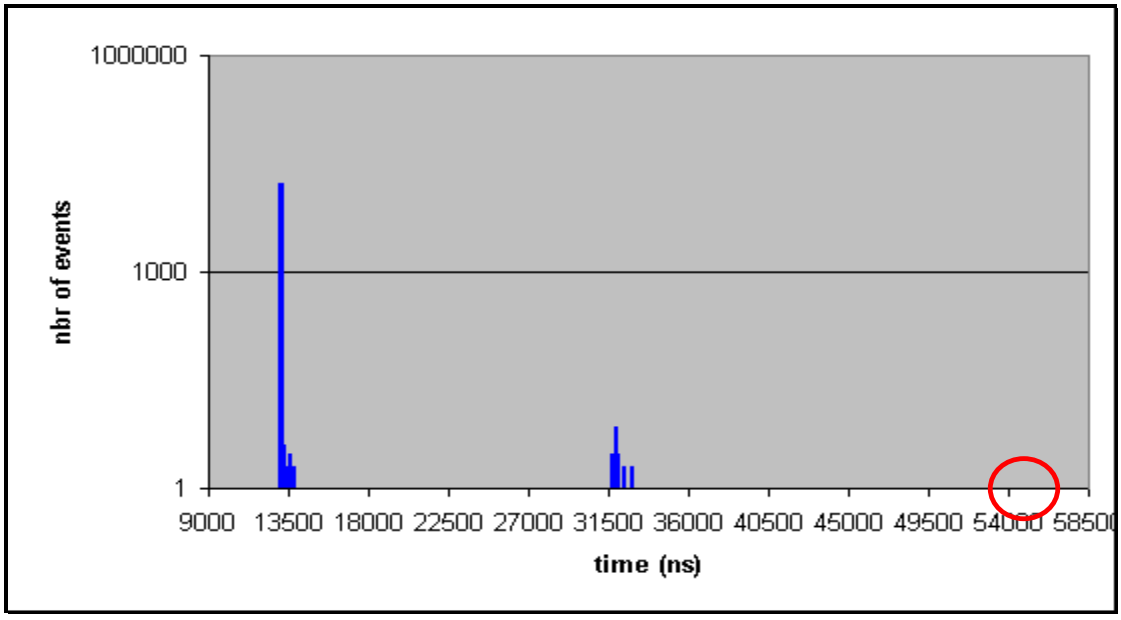

Figure 4.5-10 TSL-a-10.a – cache disabled - frequency distribution

4.5.6 Thread switch latency – cache disabled (TSL-a-128.a)

- Number of threads: 128
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
32766	12780	59370	13024

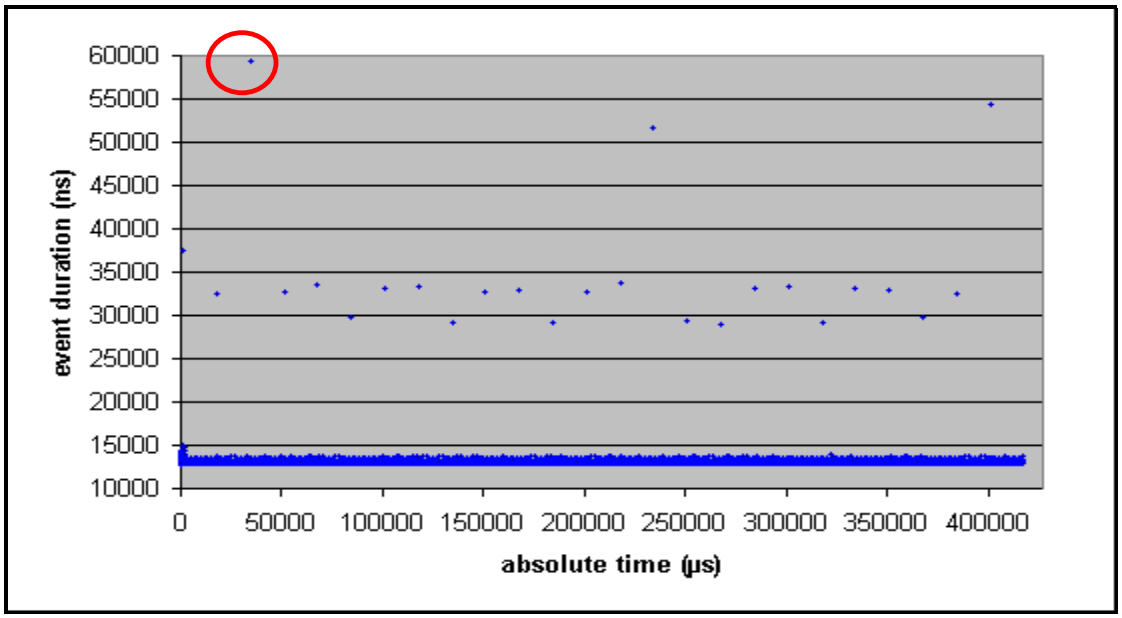

Figure 4.5-11 TSL-a-128.a – cache disabled

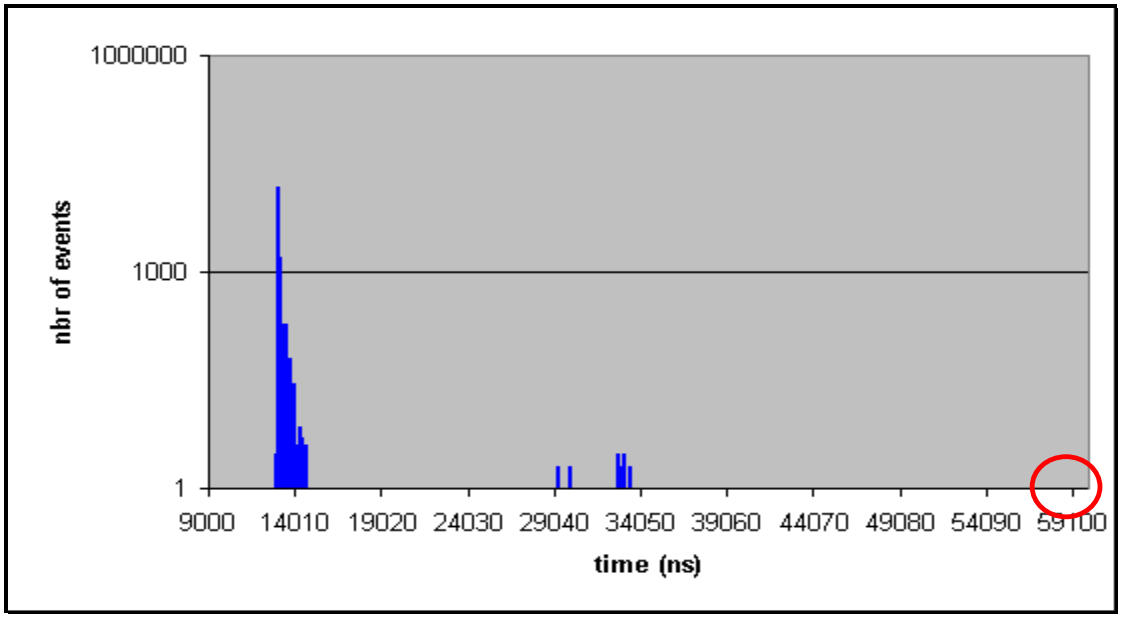

Figure 4.5-12 TSL-a-128.a – cache disabled - frequency distribution

http://www.dedicated-systems.com
email: info@dedicated-systems.com

4.6 Thread switch latency – different process

In a default configuration of VxWorks 5.3.1, all threads share the same address space. Hence, there are no different processes. If the user wants to create separate processes with their own address space, an additional component (VxVMI) needs to be purchased and installed. This component was not available to us for this evaluation so there are no test results for this series.

4.7 Binary semaphore

Sections 4.7.1 and 4.7.2 display the results for the tests on semaphore creation and deletion time. The first samples are the biggest, i.e. the first calls to create/delete a semaphore take longer than subsequent calls. This is due to caching. The reader should always keep in mind that the maximum is far more important than the average in real-time systems.

By comparing the results in sections 4.7.2 and 4.7.3, it can be seen that it doesn't take longer to delete a used semaphore than it takes to delete a semaphore that has never been used. This means that when the API call to create a semaphore is executed, the object is entirely created. Some implementations only create a minimal object at first, and defer the creation of additional resources that may be necessary until the object is first used. This is not the case in VxWorks 5.3.1.

Uncontested acquisition and release of a binary semaphore is very fast, as can be seen in sections 4.7.4 and 4.7.5. Except for the first sample in the acquisition test, the latencies were not measurable with our equipment. To perform our measurements we write traces on the PCI bus that are captured and time stamped by a PCI analyzer. The PCI bus is clocked at 33 MHz and it takes 6 cycles to write a trace on the bus. This makes that the resolution of our measurement device is 180 ns. Most of the uncontested binary semaphore acquisition and release times were smaller than 180 ns and therefore not measurable.

Section 4.7.6 displays the results for the synchronization test. In this test, we measure the time it takes the system to release a binary semaphore and schedule a higher priority thread that was thereby released. A test cycle starts with only 1 thread pending on the semaphore that is about to be released. Then, the number of threads is increased one by one until there are 255 threads of different priority pending on the semaphore, and a new test cycle can start. The idea is to investigate whether or not the time to release the semaphore (and schedule the released thread) is proportional with the number of threads waiting for the semaphore.

☺ In VxWorks 5.3.1, there is certainly no such dependency, as can be seen in Figure 4.7-11 (p.56). While the number of threads pending on the semaphore ramps up from 1 to 255, the time to release the semaphore and schedule the thread remains fairly constant (between 3 µs and 4 µs).

However, we reprocessed the test results to find out how long it takes for a thread to acquire a semaphore that is not available. When a thread acquires a semaphore that is not available, the thread needs to be added to the semaphore's queue of waiting threads. Good RTOS design requires this queue to be sorted at all times in order to keep the release time of a semaphore constant (as described in the previous paragraph). The structure of this queue is therefore very important, in order to keep the sorting time as constant and as short as possible. This did not happen in VxWorks 5.3.1 as can be seen from Figure 4.7-13 and Figure 4.7-14 (p.57). It is clear that the time it takes to add a thread to the semaphore's queue (and sort it) is proportional to the number of threads already in the queue. Queue structures that lead to better (and more constant) sorting latencies are available but require more memory, which may be unacceptable for systems with tight memory constraints.

4.7.1 Creation (SEO-a-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8192	6210	11340	6359.3

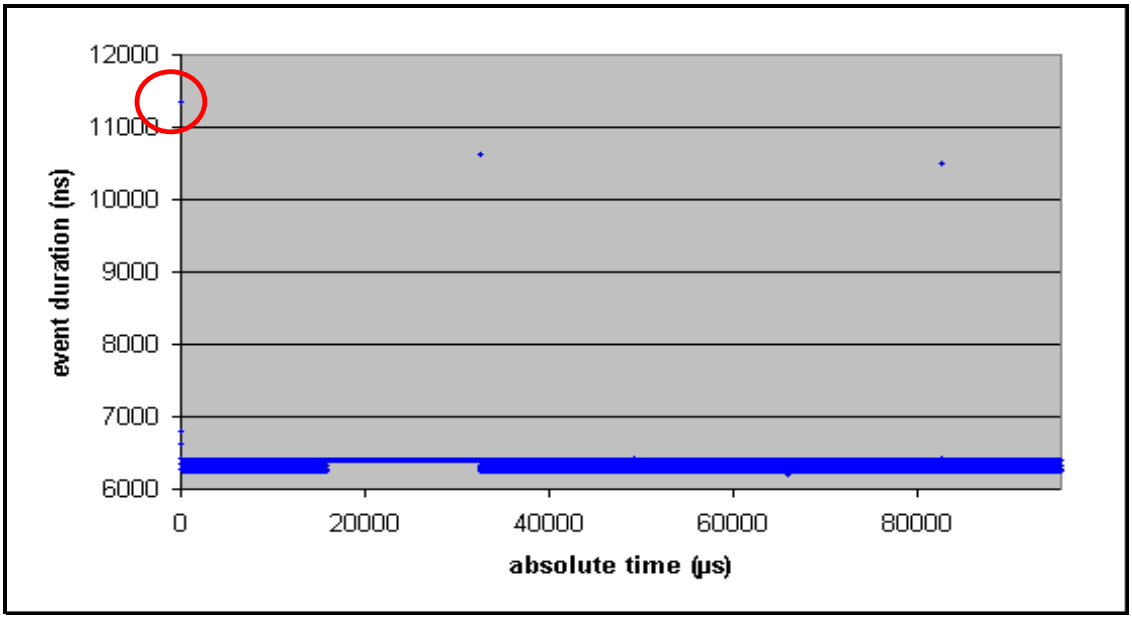

Figure 4.7-1 SEO-a-1.a

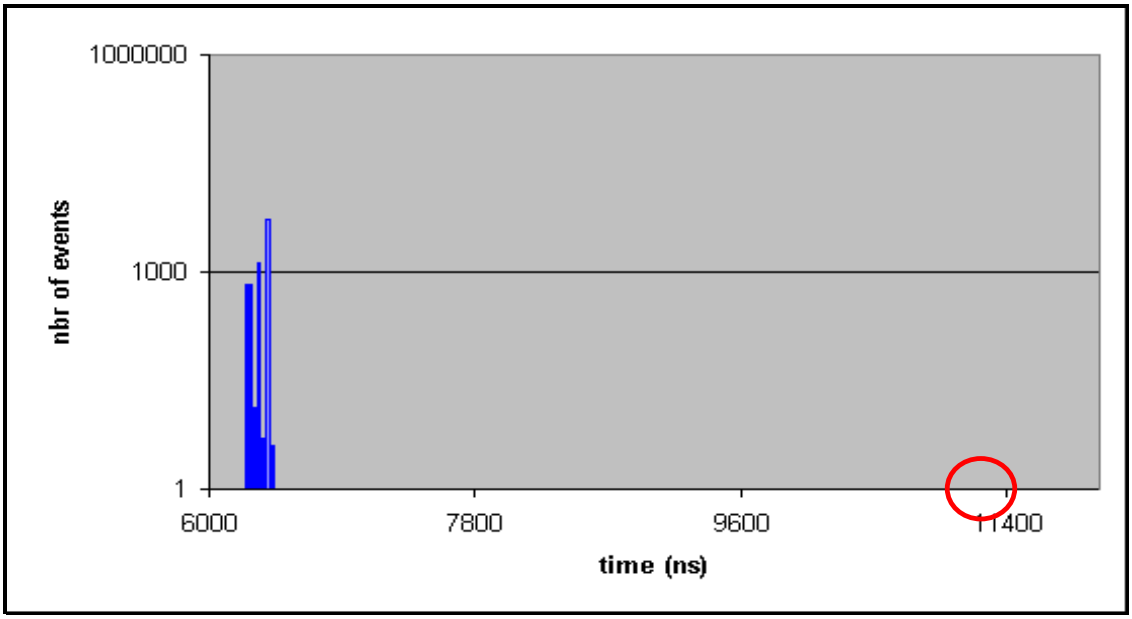

Figure 4.7-2 SEO-a-1.a - frequency distribution

4.7.2 Deletion (SEO-b-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	4890	23370	4966.8

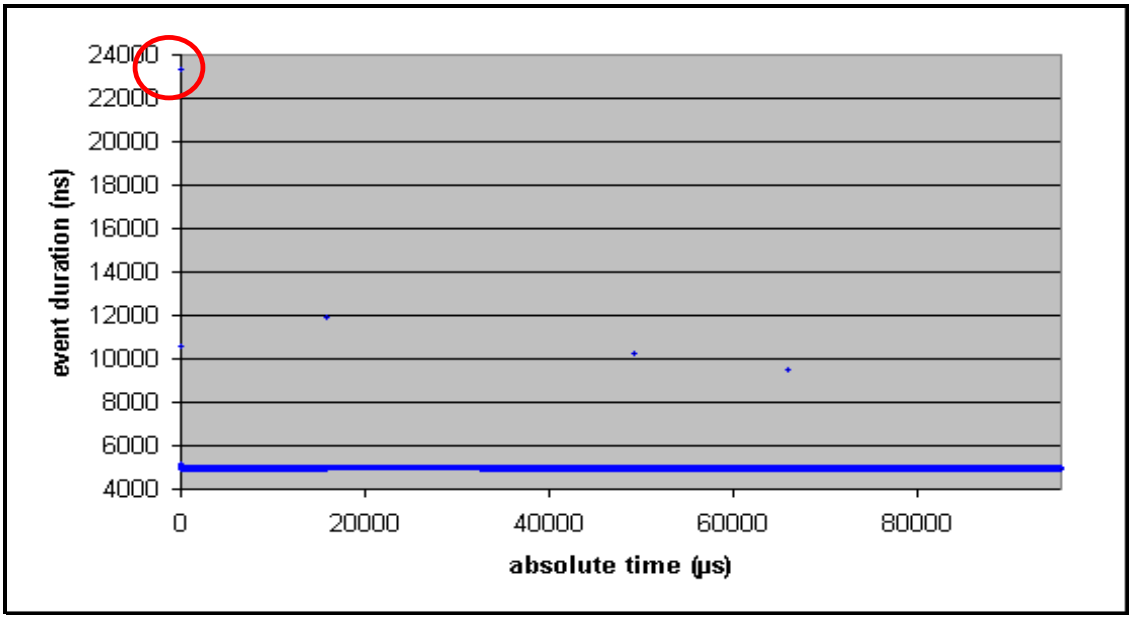

Figure 4.7-3 SEO-b-1.a

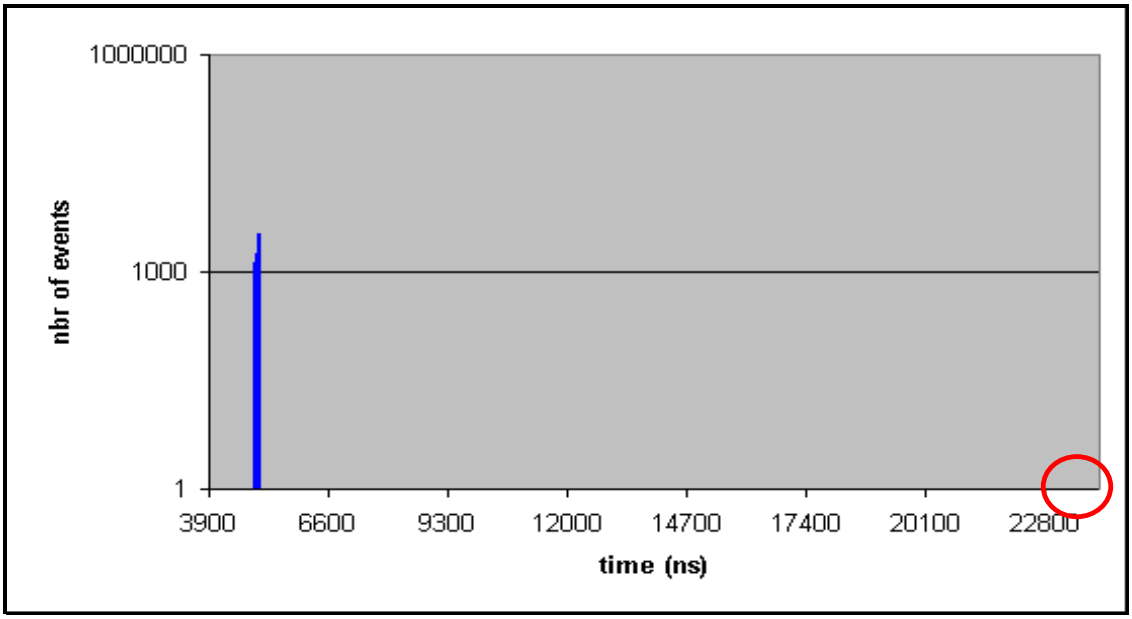

Figure 4.7-4 SEO-b-1.a - frequency distribution

4.7.3 Deletion - after use (SEO-c-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	5010	11100	5082.5

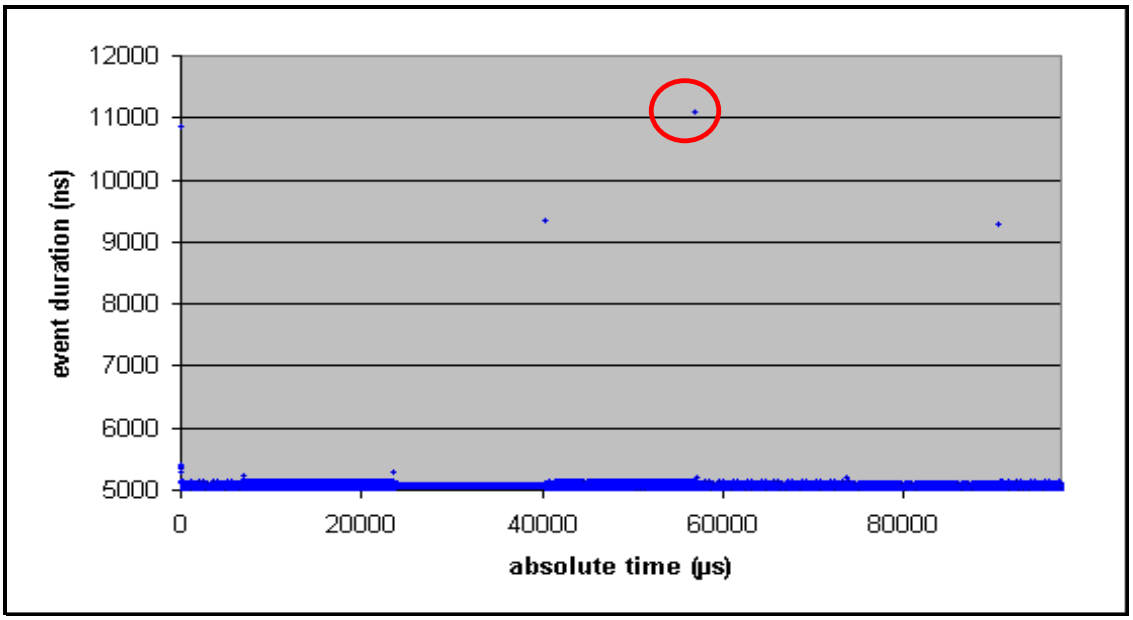

Figure 4.7-5 SEO-c-1.a

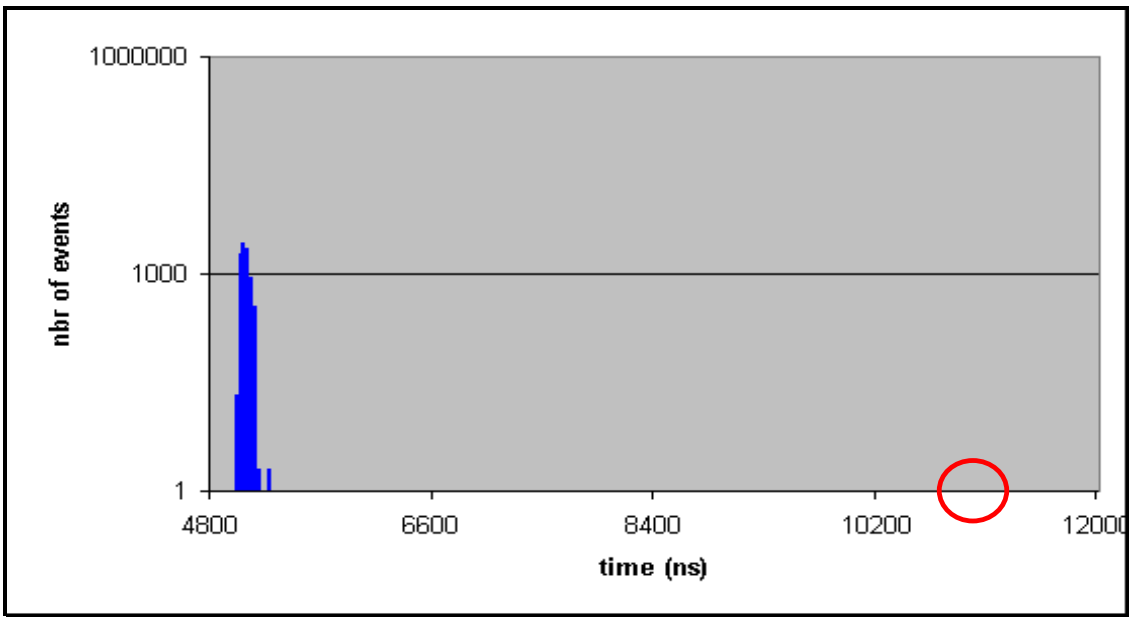

Figure 4.7-6 SEO-c-1.a - frequency distribution

4.7.4 No contention – Acquire (SEO-d-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8192	180	600	180.1

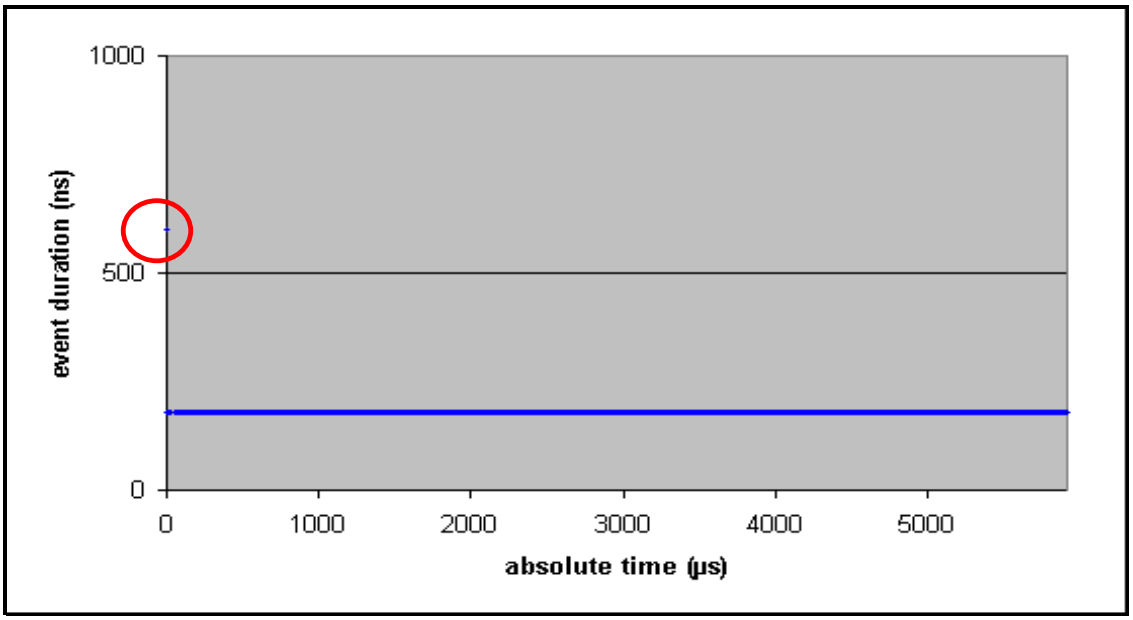

Figure 4.7-7 SEO-d-1.a

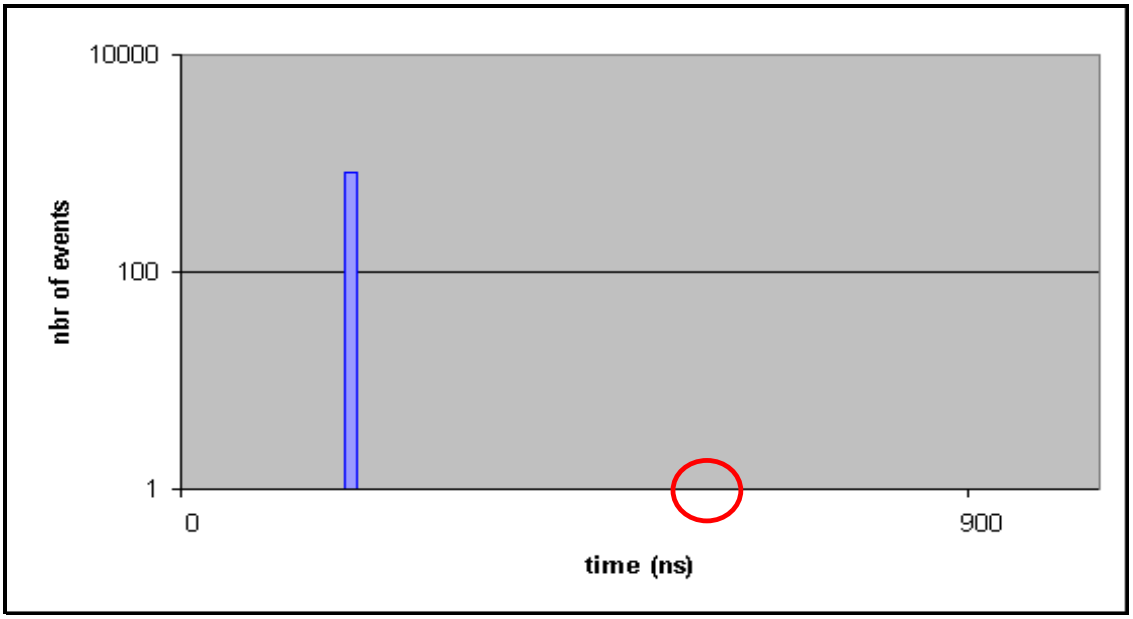

Figure 4.7-8 SEO-d-1.a - frequency distribution

4.7.5 No contention – Release (SEO-e-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	180	240	180.0

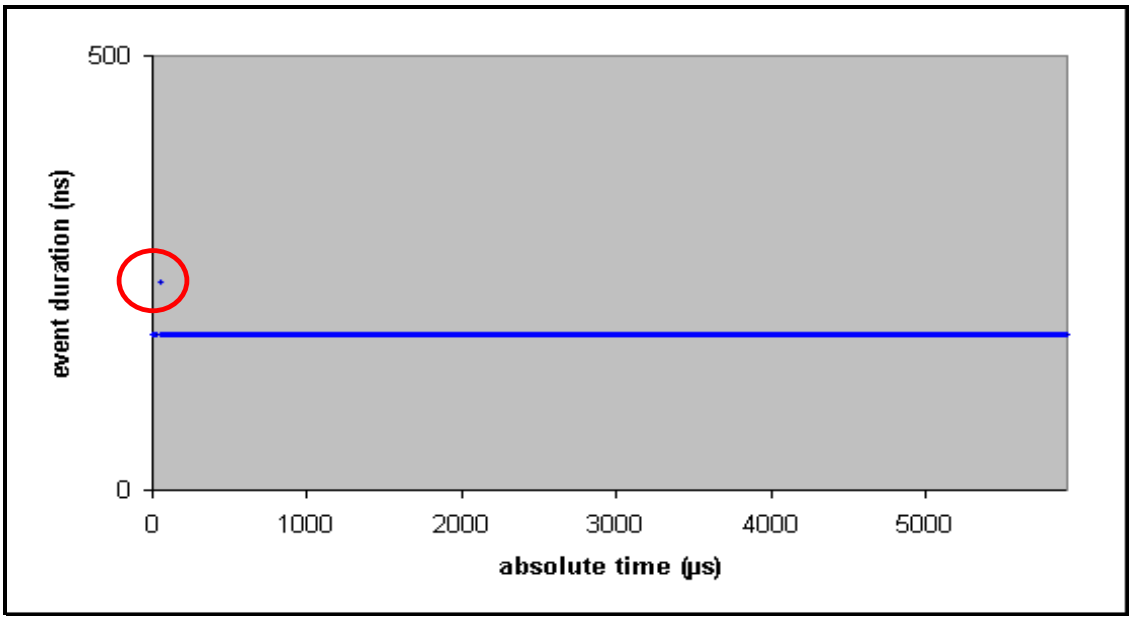

Figure 4.7-9 SEO-e-1.a

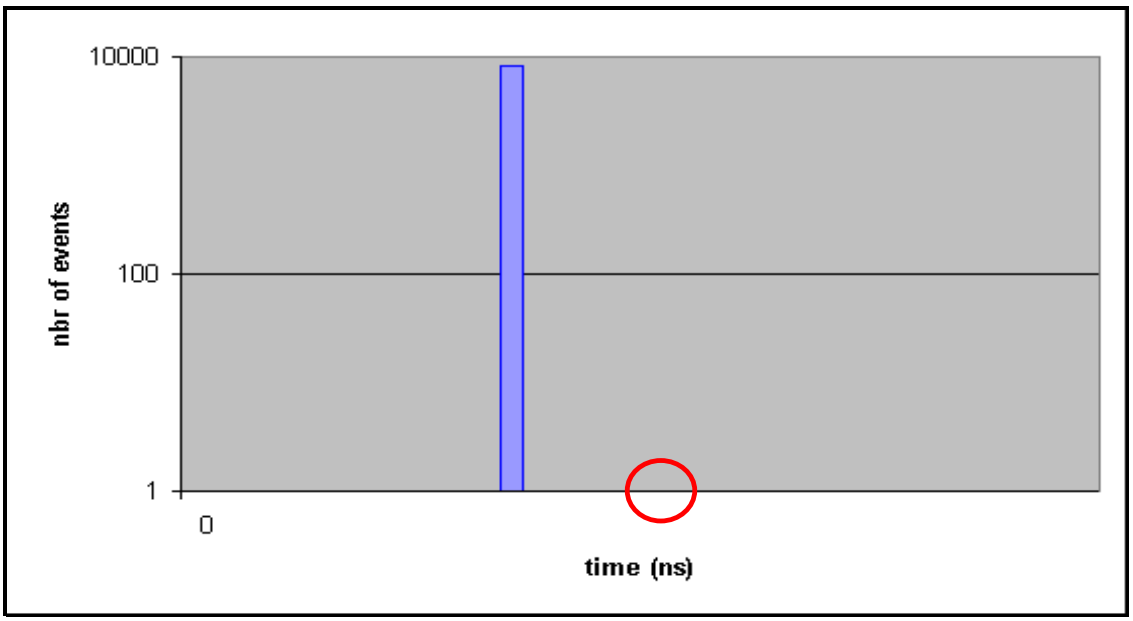

Figure 4.7-10 SEO-e-1.a - frequency distribution

4.7.6 Synchronization (SEO-f-max.a)

- Number of threads: 255
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
10866	3000	8940	3343.1

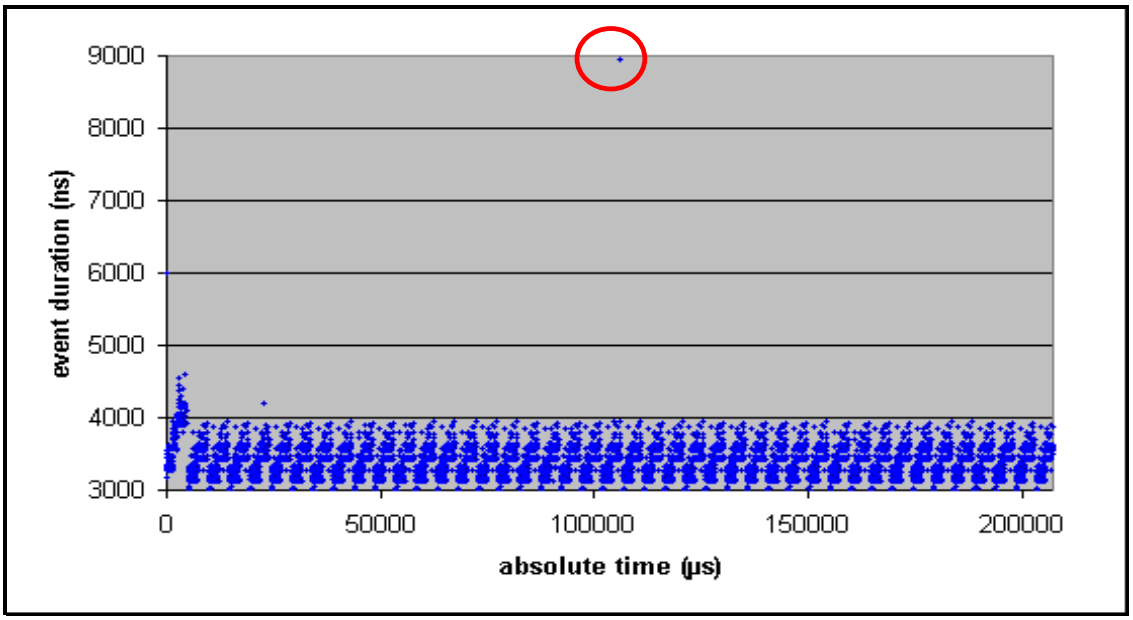

Figure 4.7-11 SEO-f-max.a

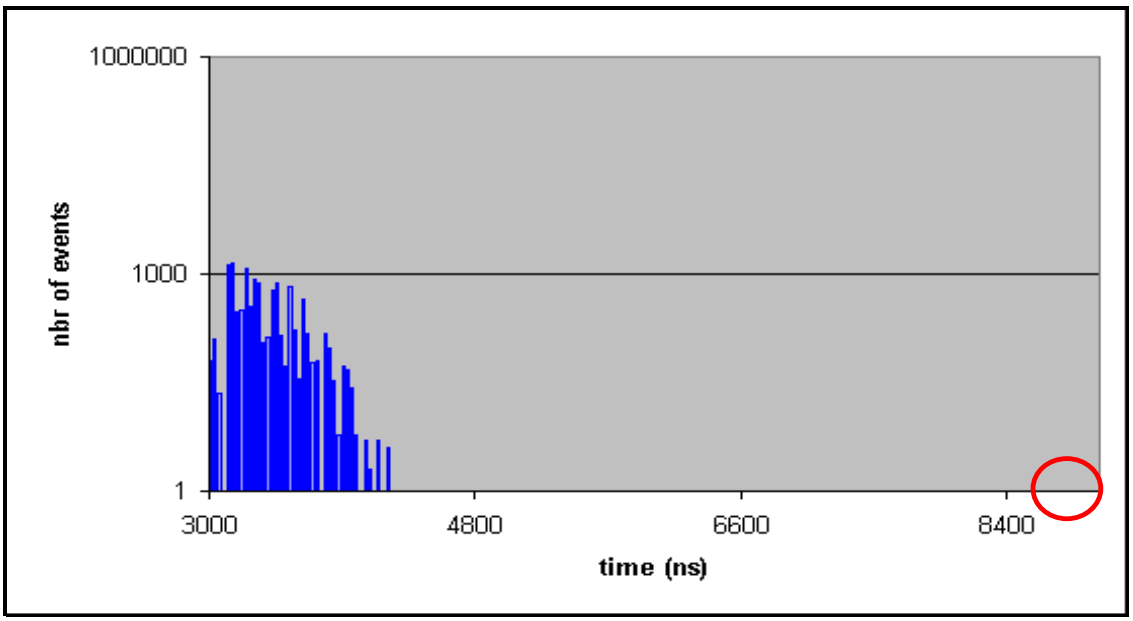

Figure 4.7-12 SEO-f-max.a - frequency distribution

4.7.7 Acquiring a semaphore that is not available

- Number of threads: 255
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
10908	3720	59850	10567.9

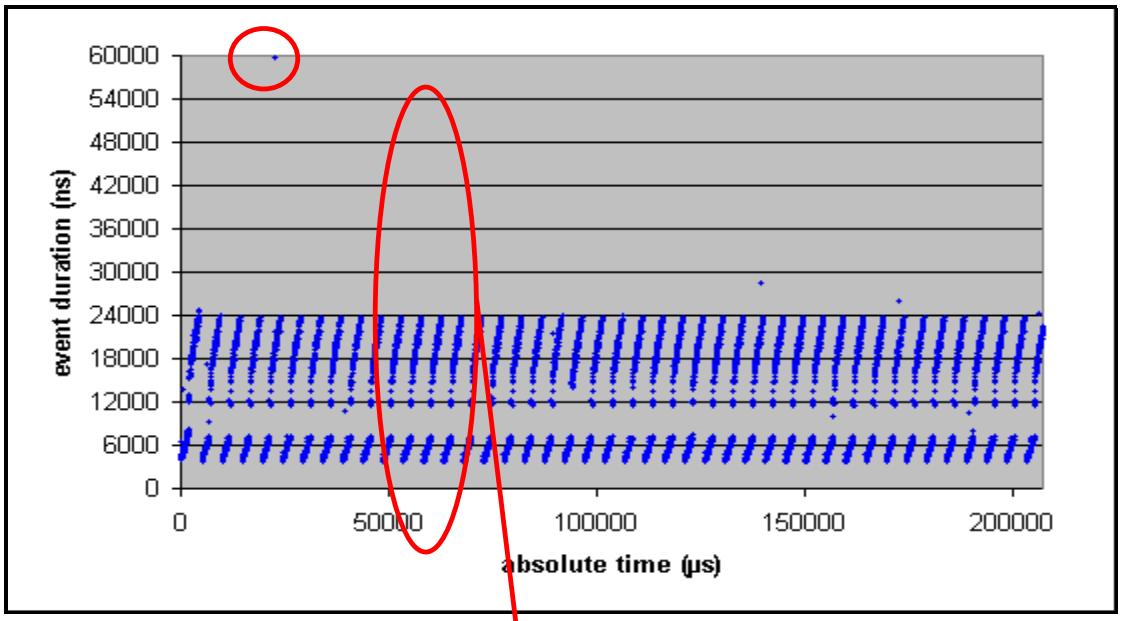

Figure 4.7-13 Acquiring a semaphore that is not available

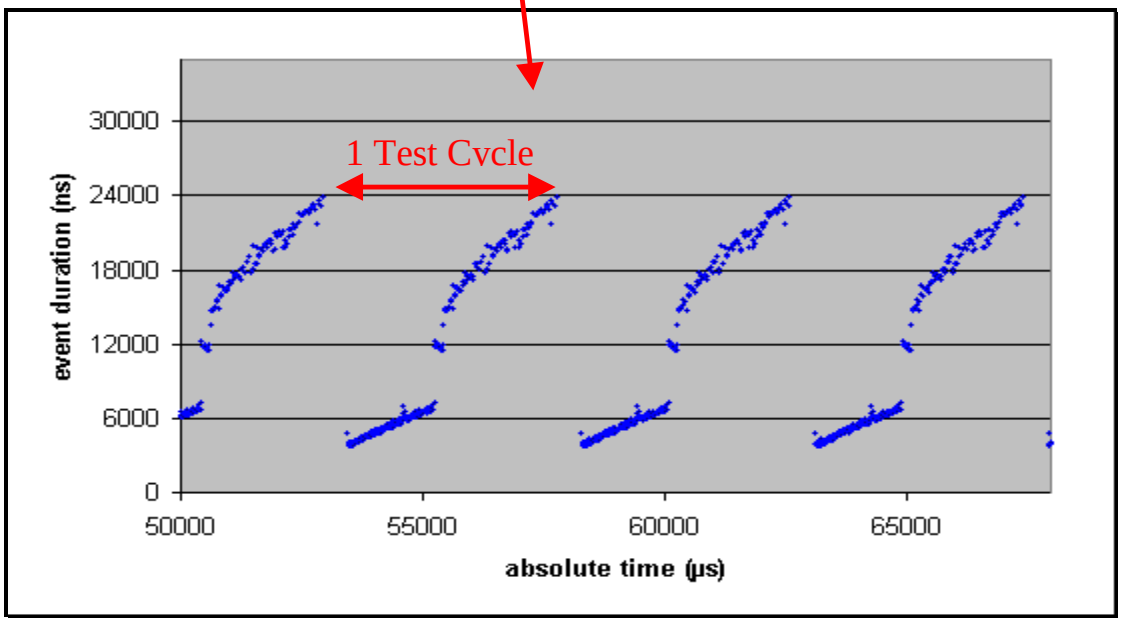

Figure 4.7-14 Acquiring a semaphore that is not available – exploded view

http://www.dedicated-systems.com
email: info@dedicated-systems.com

4.8 Counting semaphore

VxWorks 5.3.1 provides a counting semaphore, as well as a binary semaphore. A lot of vendors only provide a counting semaphore. Indeed, a binary semaphore can be created by using a counting semaphore with a maximum count of 1. However, binary semaphores are usually optimized and will therefore in general be faster than counting semaphores with a maximum count of 1.

In this section, the tests on uncontested acquisition and release were repeated with a counting semaphore, since the results were not measurable when executed with a binary semaphore. The same results were obtained with counting semaphores : the average uncontested acquisition and release time is smaller than the resolution of our measurement device (180 ns).

4.8.1 No contention – Acquire (SEO-d-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8192	180	750	180.1

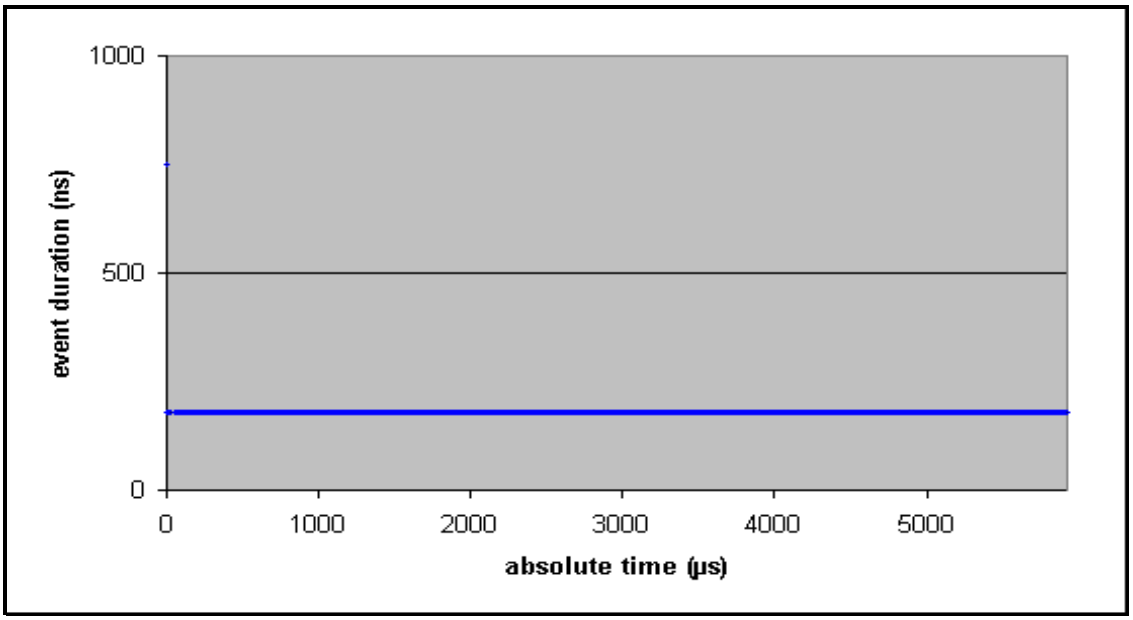

Figure 4.8-1 SEO-d-1.a

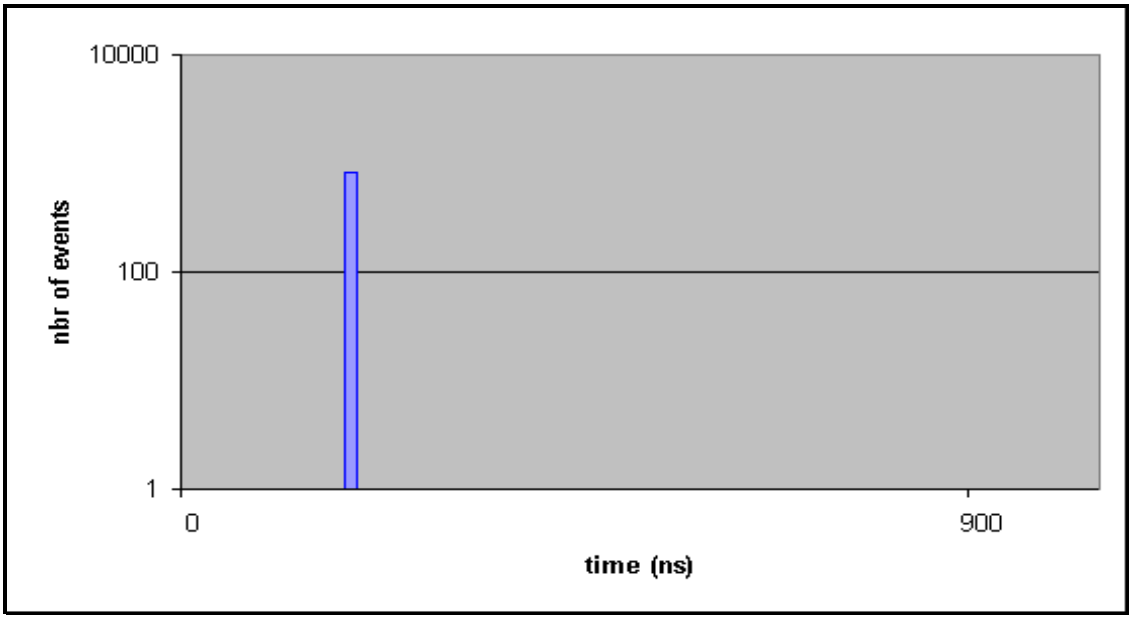

Figure 4.8-2 SEO-d-1.a - frequency distribution

4.8.2 No contention – Release (SEO-e-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	180	18030	183.1

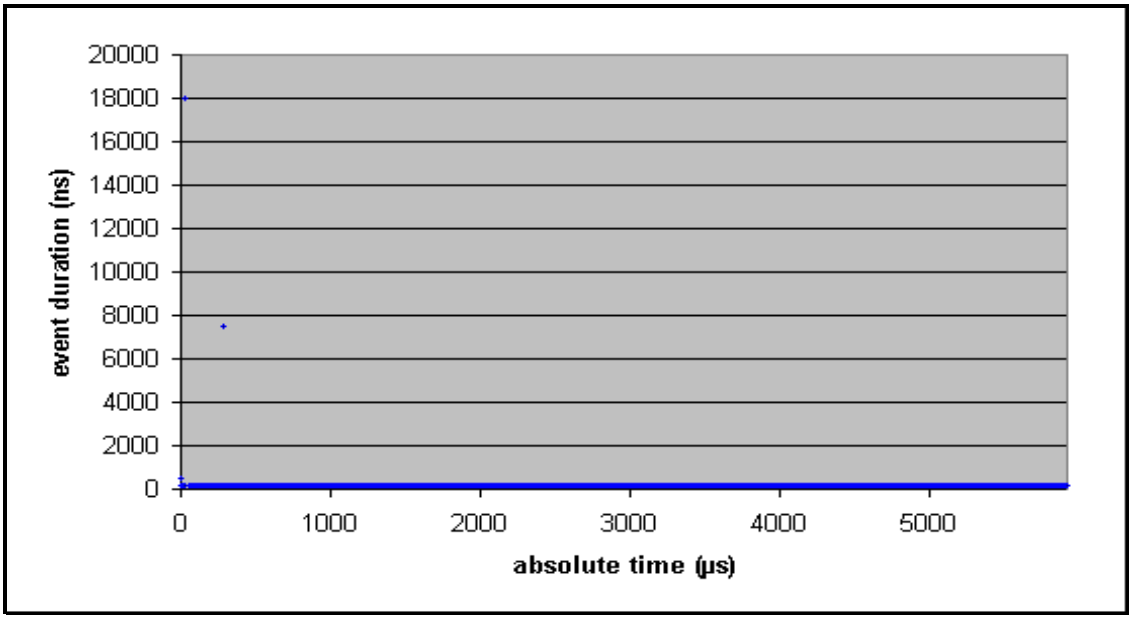

Figure 4.8-3 SEO-e-1.a

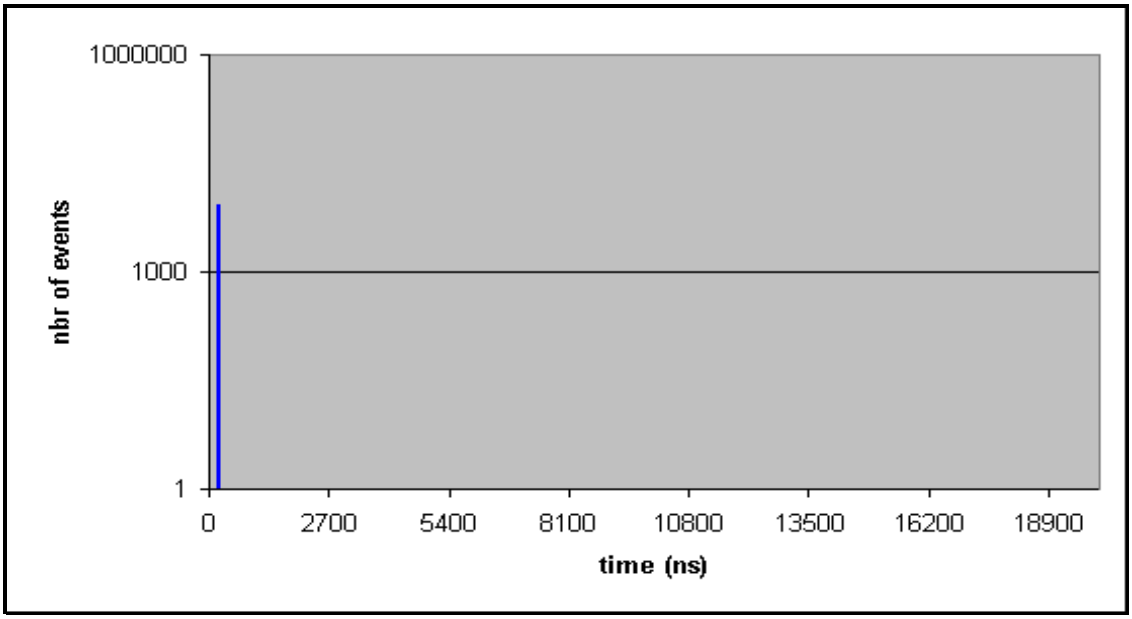

Figure 4.8-4 SEO-e-1.a - frequency distribution

4.9 Mutex

Sections 4.9.1 and 4.9.2 display the results for uncontested mutex release and acquisition times. Just as with semaphores, the average uncontested mutex acquisition/release time is smaller than 180 ns and therefore not measurable.

VxWorks 5.3.1 has a priority inheritance mechanism, which is absolutely crucial for an RTOS to have. We tested this by creating a situation with 3 threads where the priority inversion problem occurs: a high priority thread wants to acquire a mutex that is owned by a low priority thread. A medium priority thread keeps the low priority thread from running and releasing the mutex so that it can't be acquired by the high priority thread. In this test, we measured the time it takes for the highest priority thread to acquire the mutex. That time includes the time it takes to boost the priority of the lowest priority thread, have it release the mutex, and switch back to the highest priority thread so it can acquire the mutex. The priority inheritance mechanism in VxWorks 5.3.1 seems stable, the average time it took to perform this series of actions was 10.2 µs, with a maximum of 33 µs (due to the clock interrupt) as can be seen in Figure 4.9-5 (p. 64).

The reader should be aware that this the smallest possible time interval to handle priority inversion. In a real situation the low priority thread may perform additional work before releasing the mutex.

An interesting observation to make in the results of the priority inversion test are the peaks caused by the clock interrupt. While the average latency is 10.2 µs as mentioned above, a lot of samples hover around 15 µs. These samples occur every 16.7 ms when the clock ticks. From this test, as well as from previous tests, it seems that it takes about 4 µs on the average to service the clock interrupt (on a Pentium 200 Mhz). However, in this test, the maximum latency (also caused by the clock interrupt) is 33 µs. Apparently, in this incident, it took almost 23 µs to service the clock interrupt.

For a detailed discussion on the execution time of the clock ISR and how this may impact the performance, the reader is referred to section 0.

4.9.1 No contention – Acquire (SEO-d-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8192	180	330	180.0

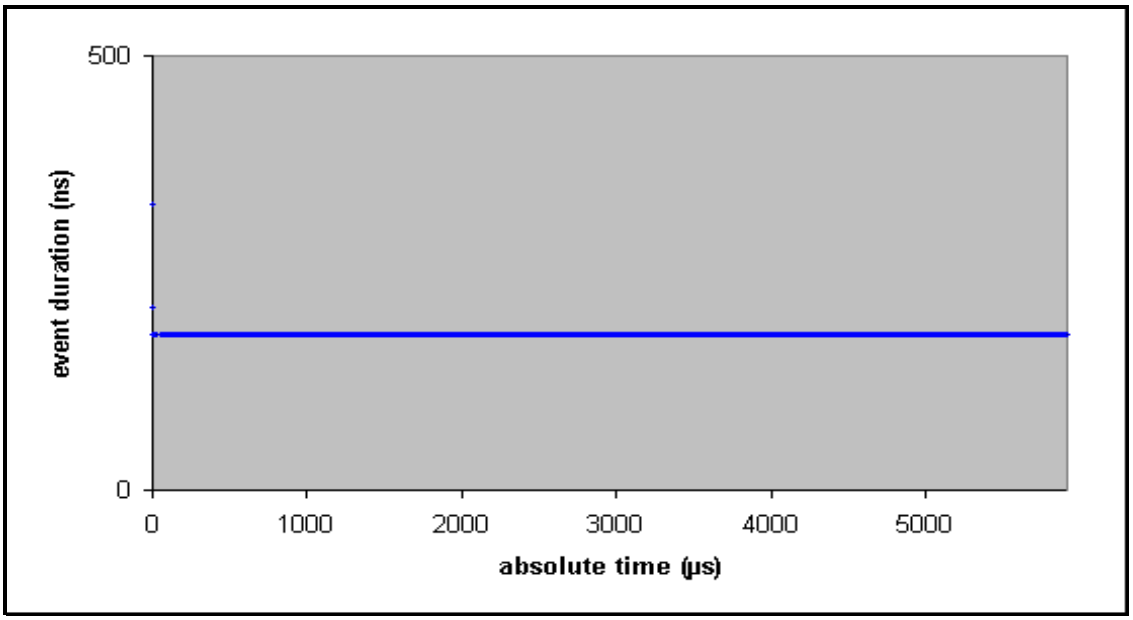

Figure 4.9-1 SEO-d-1.a

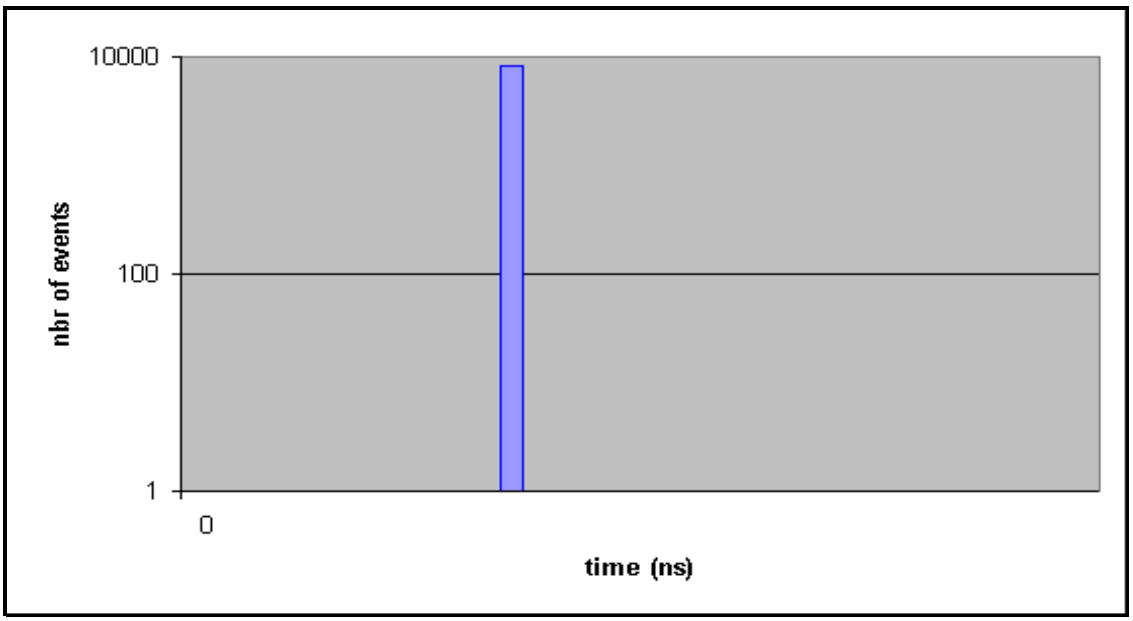

Figure 4.9-2 SEO-d-1.a - frequency distribution

4.9.2 No contention – Release (SEO-e-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
8191	180	18210	182.2

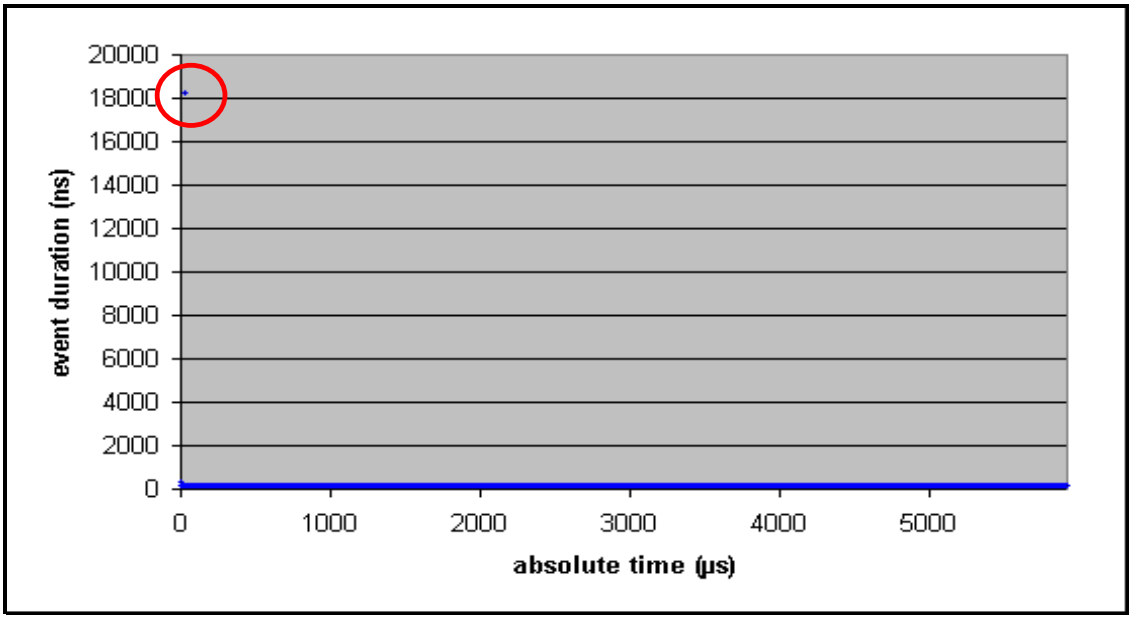

Figure 4.9-3 SEO-e-1.a

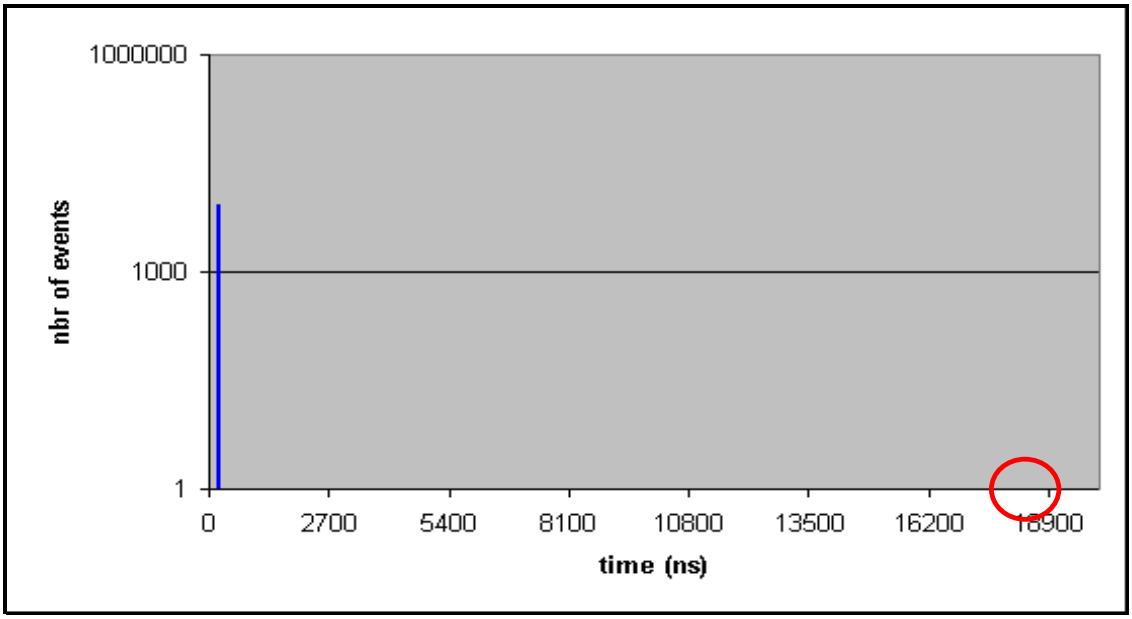

Figure 4.9-4 SEO-e-1.a - frequency distribution

4.9.3 Priority inversion (SEO-g-3.a)

- Number of threads: 3
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	10260	32940	10293.1

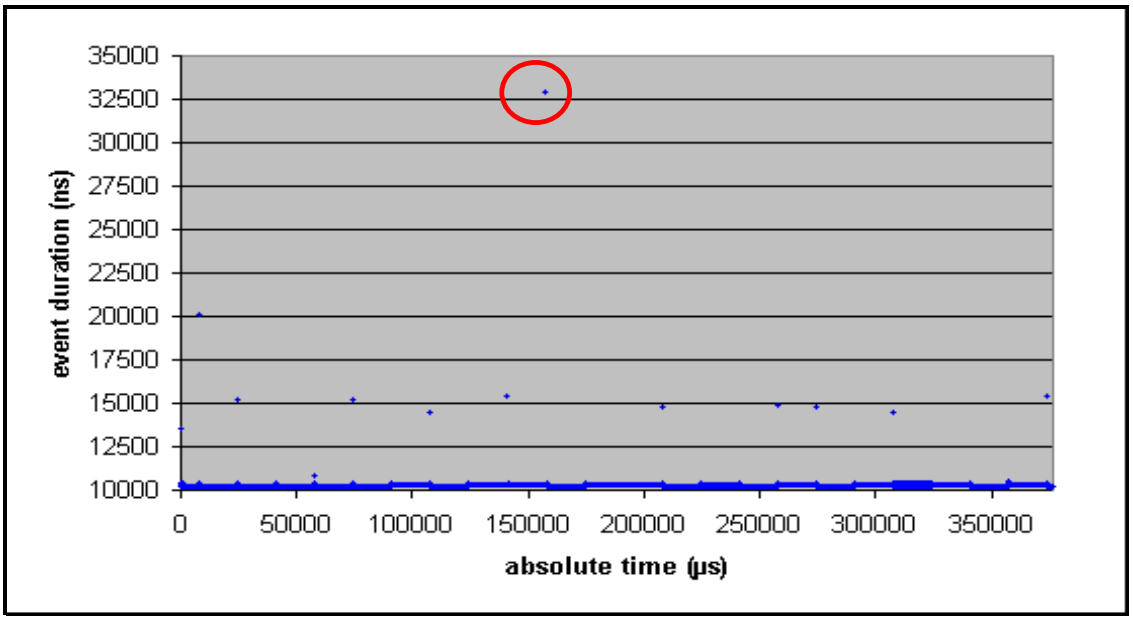

Figure 4.9-5 SEO-g-3.a

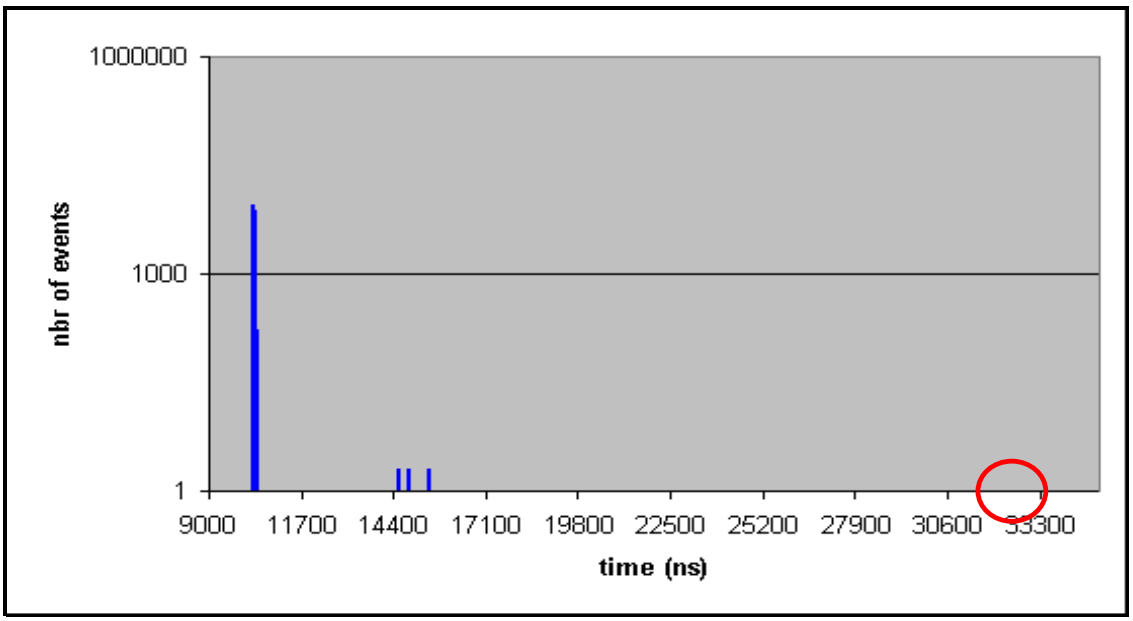

Figure 4.9-6 SEO-g-3.a - frequency distribution

http://www.dedicated-systems.com
email: info@dedicated-systems.com

4.10 File system – proprietary

VxWorks 5.3.1 does not have a proprietary file system. The local file systems supported are dosFS (an MS-DOS compatible file system), an rt11Fs compatible file system, rawFs (a raw file system), tapeFs (a tape file system) and TFFS (a flash file system). For the tests on dosFs, the reader is referred to section 4.11.

The vendor has informed us that a CDROM file system will be bundled with Tornado II.

4.11 File system – DOS

The tests in this section evaluate the performance of the MS-DOS compatible file system dosFs.

Diskettes formatted using the dosFs file system are compatible with MS-DOS diskettes up to and including release 6.2. Hard disks initialized by the two different file systems have slightly different formats, but the data is compatible and dosFs can be configured to use a disk formatted by MS-DOS. DosFs provides additional functionality like longer filenames and contiguous or non-contiguous files on a per-file basis. This feature is to satisfy demand of real-time applications: while non-contiguous files result in a more efficient use of disk space, contiguous files offer better performance.

We first measured the time it takes to create and delete a file. The average to create was 2.7 ms while the maximum was over 86 ms (Figure 4.11-1). The average time to delete a file was 3.1 ms with a peak of 4 ms.

Three tests were performed for reading and writing to a file. The difference between the tests being the size of data transferred: 1 byte, 1 block of 512 bytes (1b), or 10 blocks (5120 bytes, 10b). Contiguous files were used since they are more appropriate for the use in real-time systems. There is always a peak at the start of the tests that read from a file. This is due to the fact that the data is cached in memory when it is first read from disk, i.e. subsequent reads are read from memory instead of disk. It was not possible to disable the read-cache during the tests.

The tests for writing to a file however were implemented in such way that the data for each write operation was actually written to disk instead of cache memory. One measurement consists of a “write” call followed by a call that flushes the write buffers to make sure the data is written to disk.

Although contiguous files are used, the test results still show significant peaks which occur periodically. This is clearly demonstrated in section 4.11.7, which handles the write operations of 1 block (512 bytes, also the size of 1 sector on disk). In Figure 4.11-14 (p.73), which shows a zoom-in of the test results, the reader can observe that every 32nd write operation takes about 34 ms instead of 2 ms.

The same behavior can be observed when 10 blocks (5120 bytes) are written per write operation (section 4.11.8). As can be observed in Figure 4.11-16 (p.74), the peak (more than 40 ms high) occurs now every 2 or 3 read operations.

It seems that such peak occurs every time after 16 Kbyte are written, which is every 32 sectors on disk. In theory, this should not be due to a head movement since the file was specified to be contiguous, and no error message was returned during the test.

4.11.1 Creation (FS-a-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	2594820	86212020	2715453.2

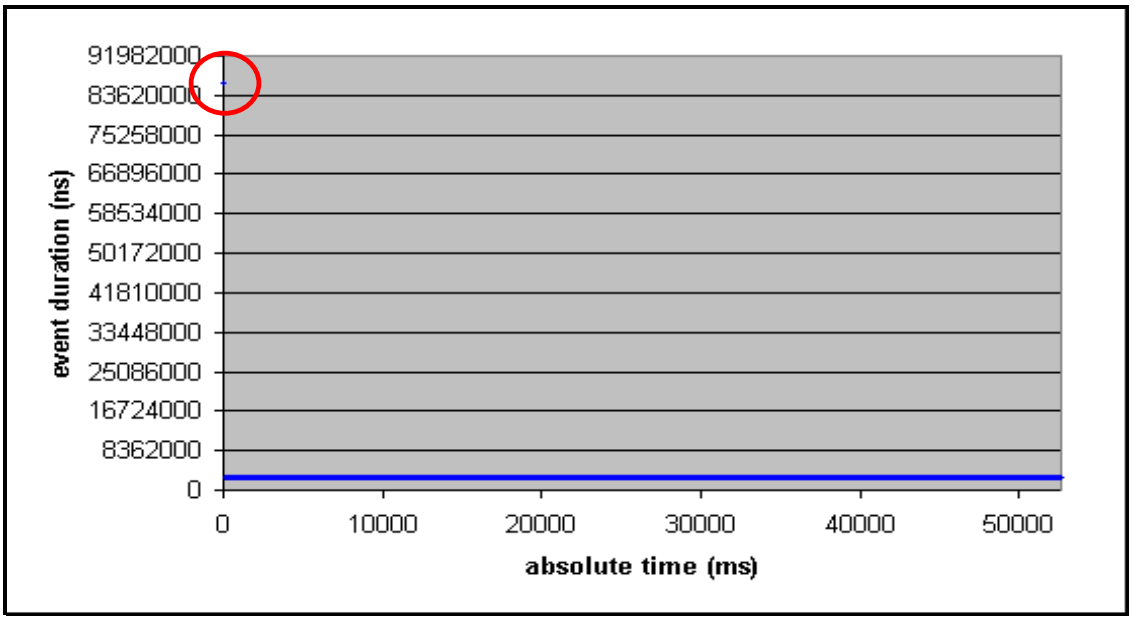

Figure 4.11-1 FS-a-1.a

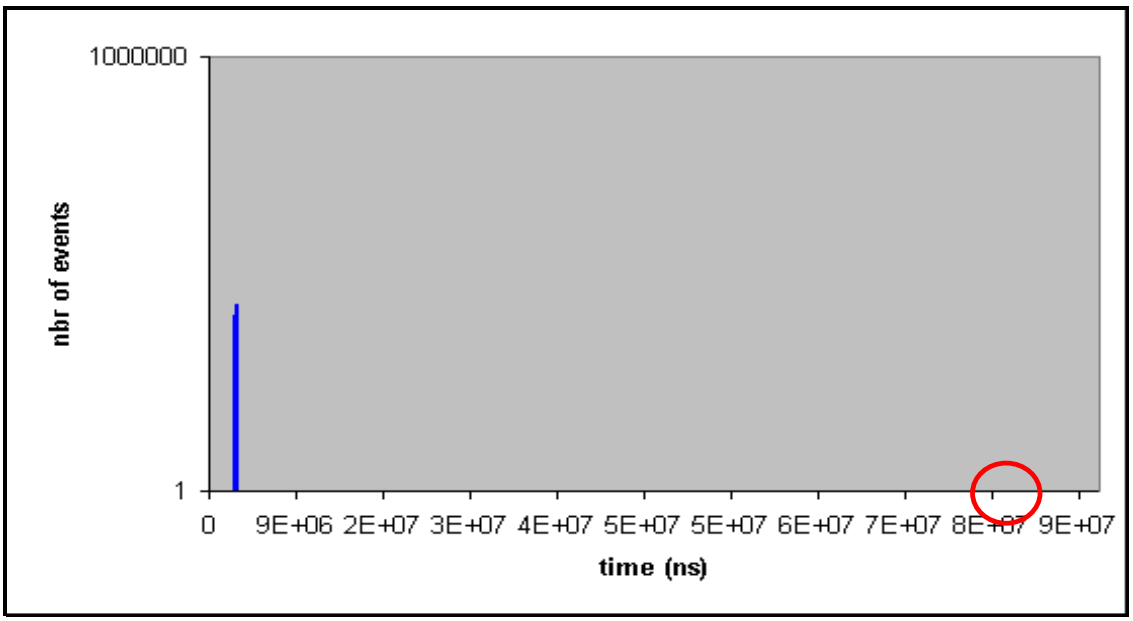

Figure 4.11-2 FS-a-1.a - frequency distribution

4.11.2 Deletion (FS-b-1.a)

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	2939310	4064520	3140348.0

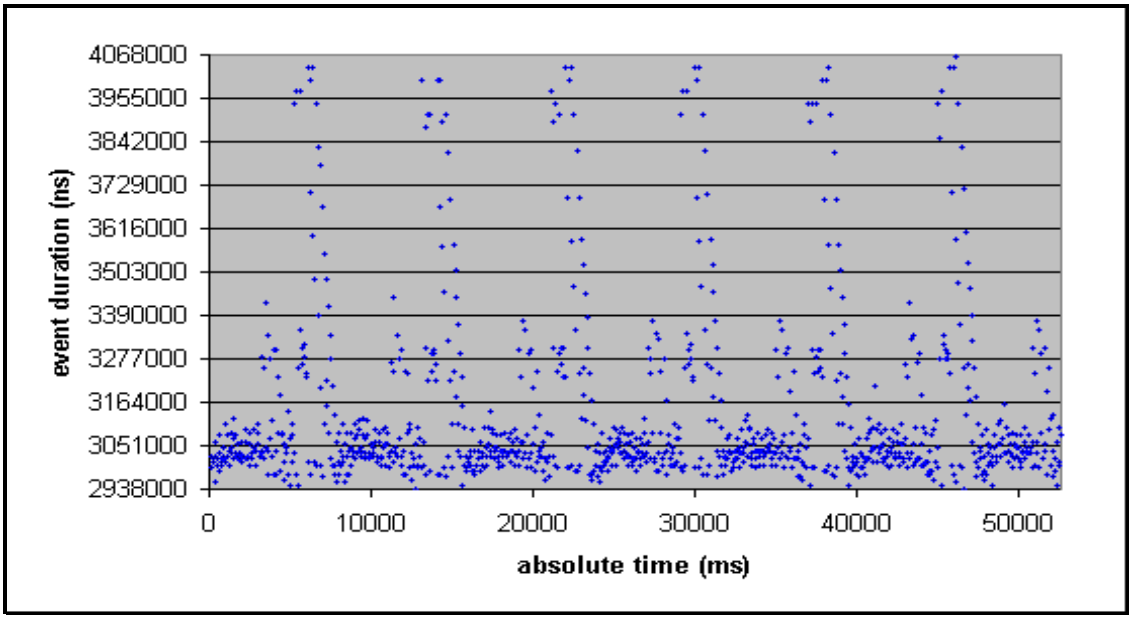

Figure 4.11-3 FS-b-1.a

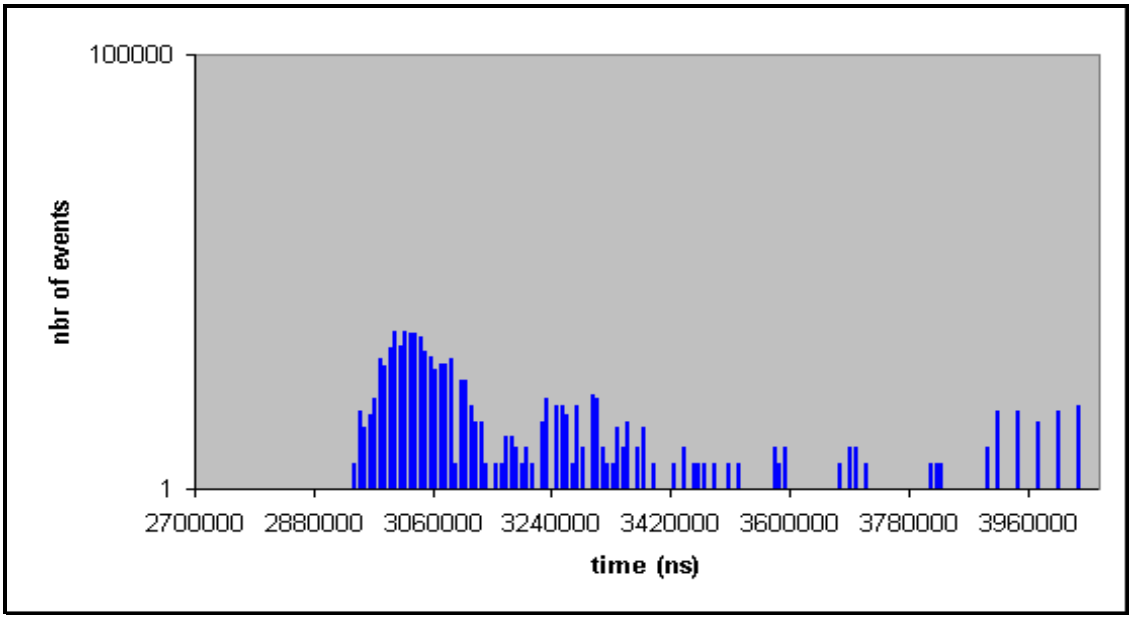

Figure 4.11-4 FS-b-1.a - frequency distribution

4.11.3 Read (FS-c-1.a.1)

- Number of threads: 1
- Memory model: a
- Number of bytes: 1

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	4440	1,13E+08	117715,1

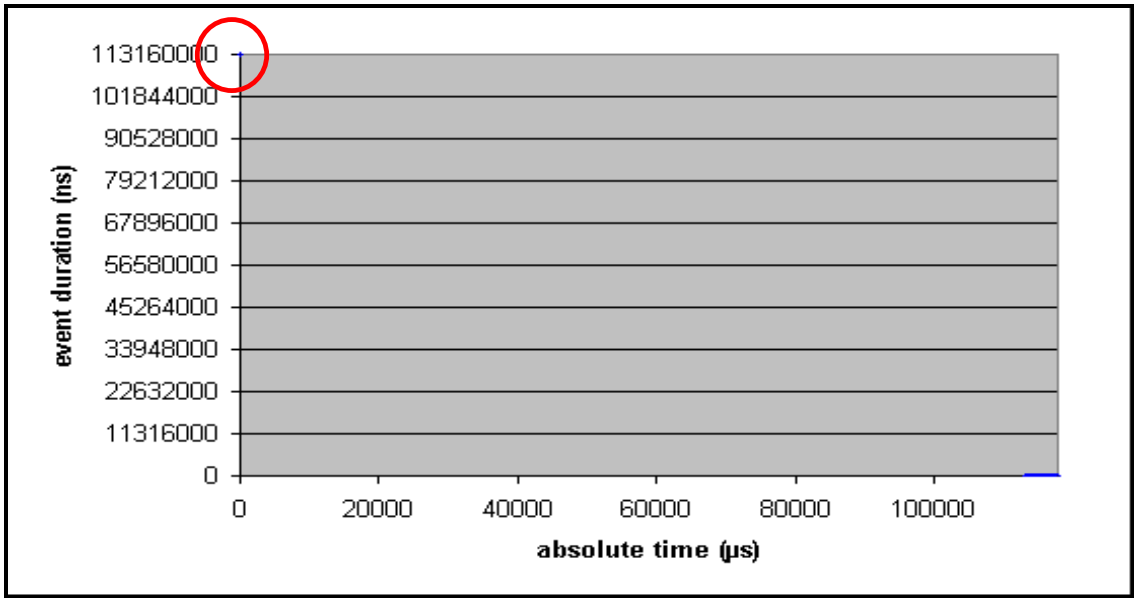

Figure 4.11-5 FS-c-1.a.1

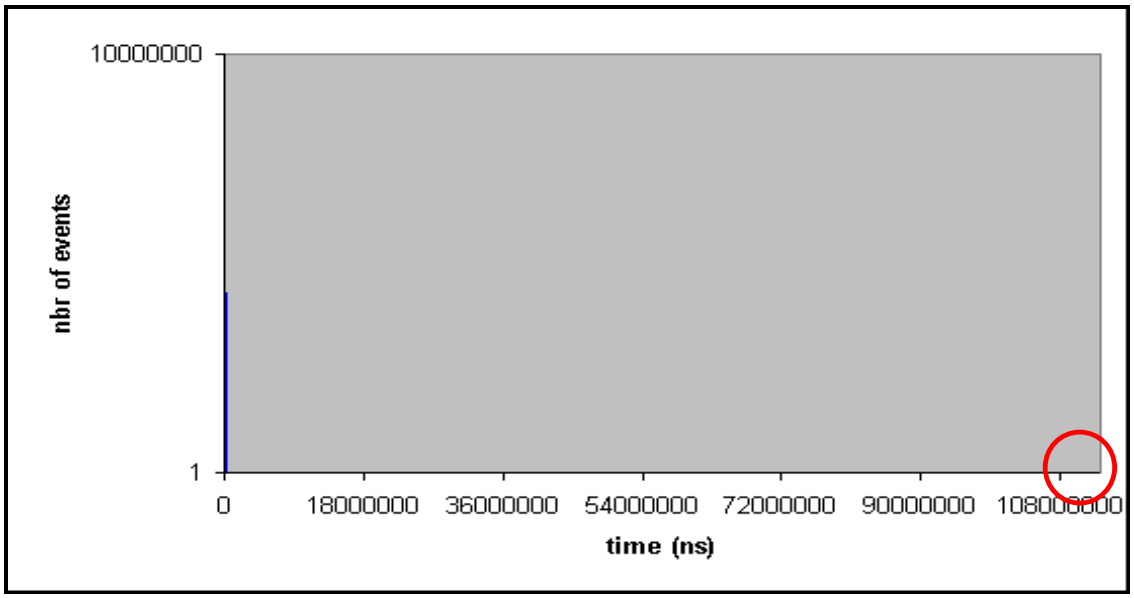

Figure 4.11-6 FS-c-1.a.1 - frequency distribution

4.11.4 Read (FS-c-1.a.1b)

- Number of threads: 1
- Memory model: a
- Number of bytes: 1 block (512 bytes)

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	9960	1,1E+08	197557,9

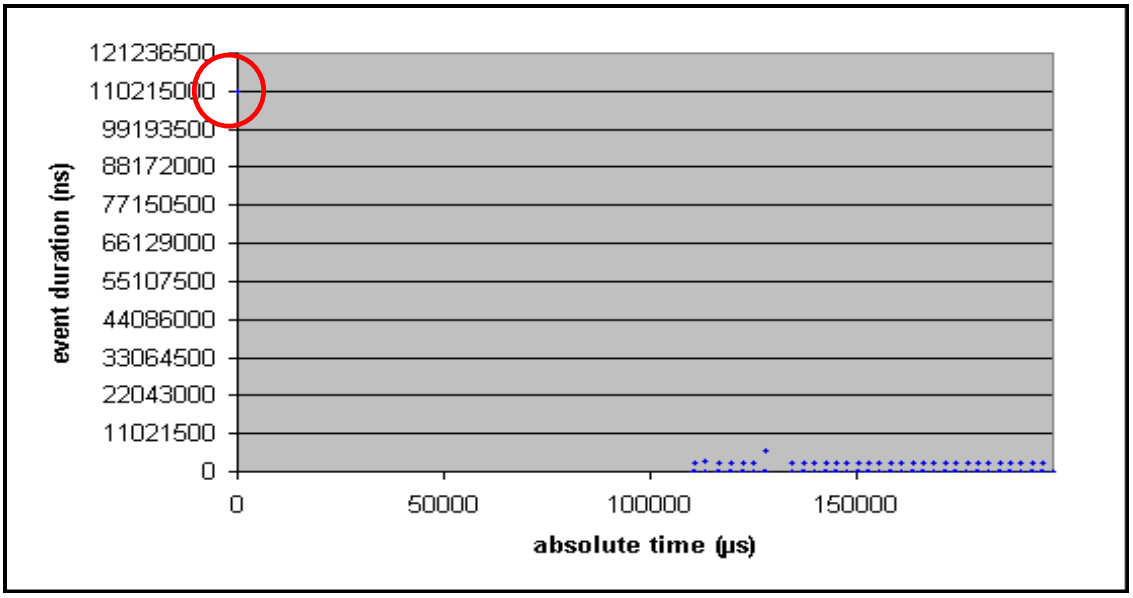

Figure 4.11-7 FS-c-1.a.1b

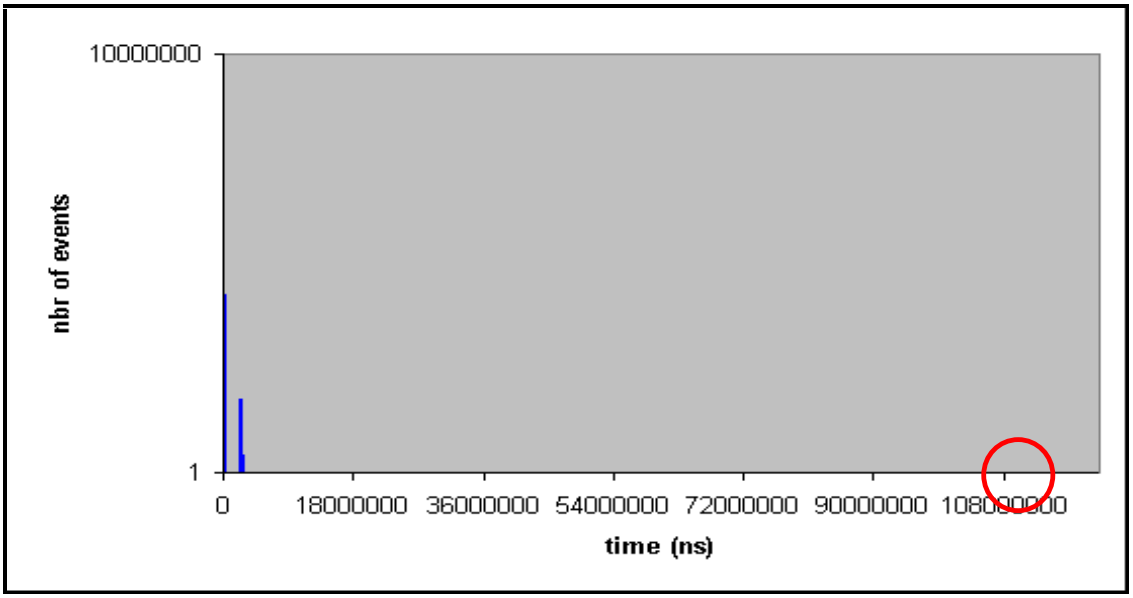

Figure 4.11-8 FS-c-1.a.1b - frequency distribution

4.11.5 Read (FS-c-1.a.10b)

- Number of threads: 1
- Memory model: a
- Number of bytes: 10 blocks (5120 bytes)

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	63660	26568000	810840,7

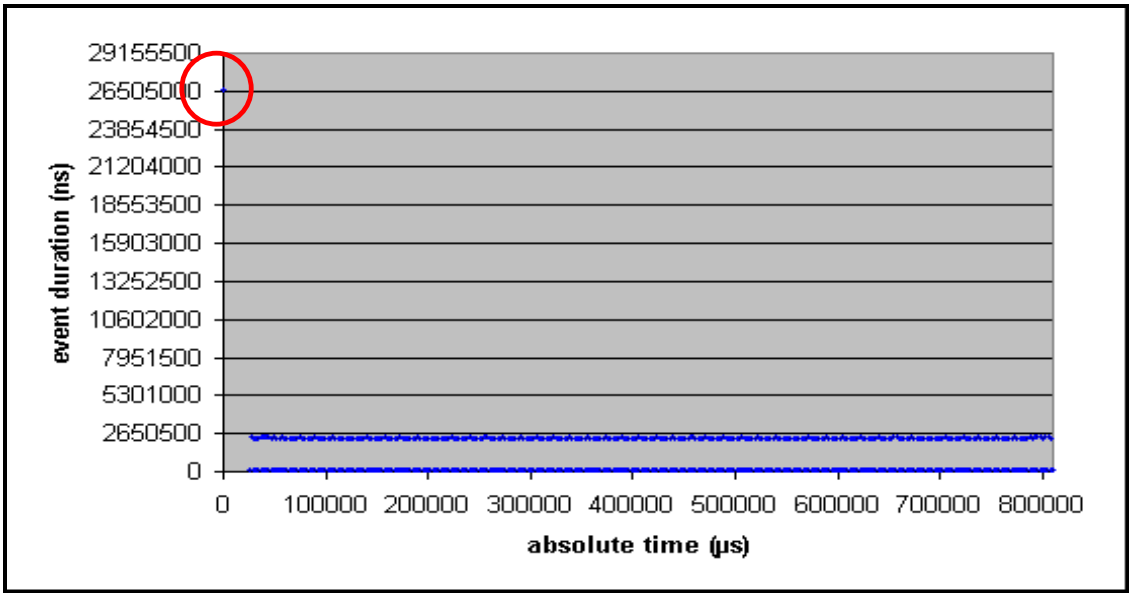

Figure 4.11-9 FS-c-1.a.10b

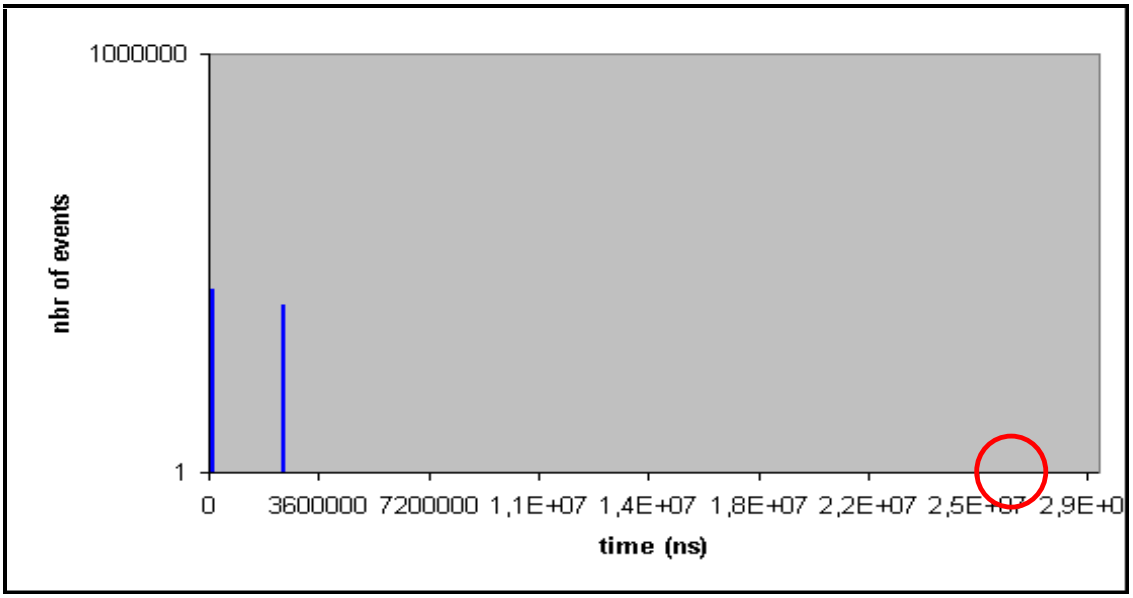

Figure 4.11-10 FS-c-1.a.10b - frequency distribution

4.11.6 Write (FS-d-1.a.1)

- Number of threads: 1
- Memory model: a
- Number of bytes: 1

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	2097300	44082000	2188209,4

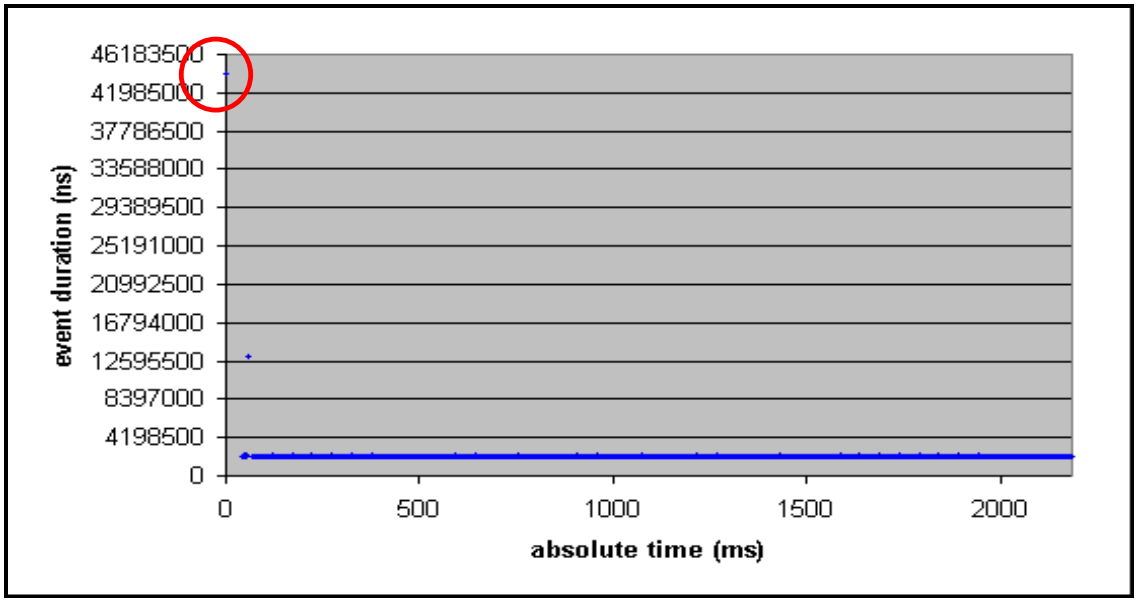

Figure 4.11-11 FS-d-1.a.1

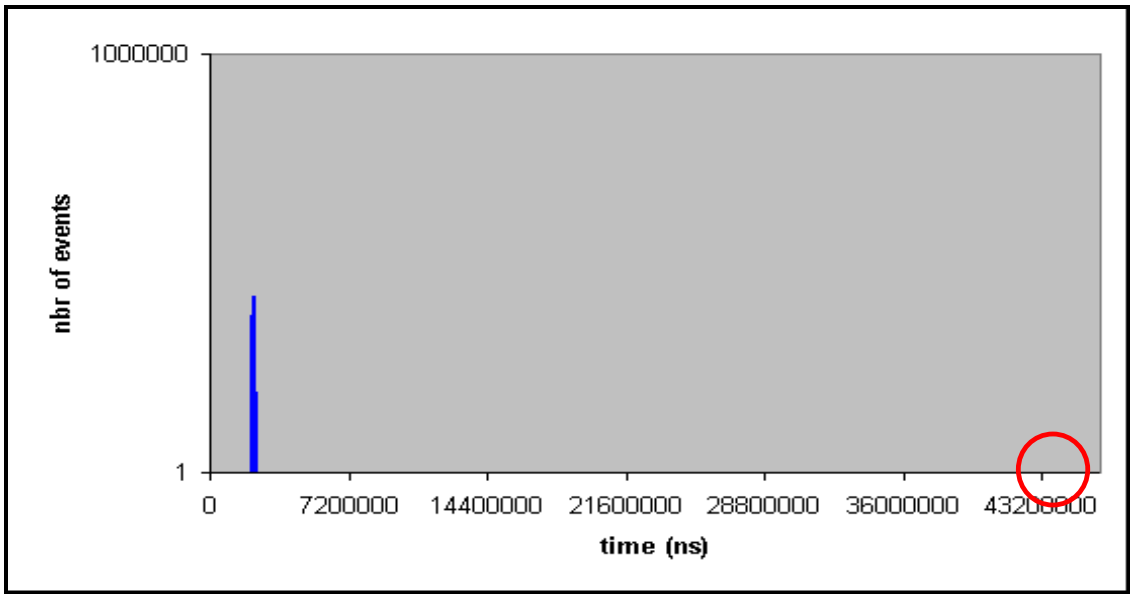

Figure 4.11-12 FS-d-1.a.1 - frequency distribution

4.11.7 Write (FS-d-1.a.1b)

- Number of threads: 1
- Memory model: a
- Number of bytes: 1 block (512 bytes)

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	2104920	38081010	3194217,8

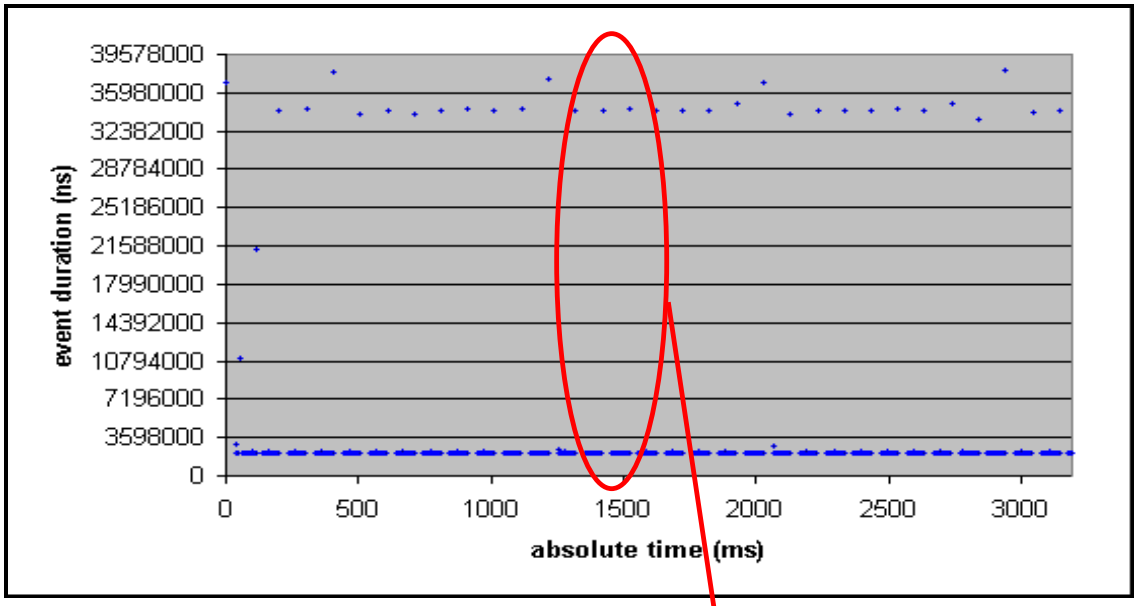

Figure 4.11-13 FS-d-1.a.1b

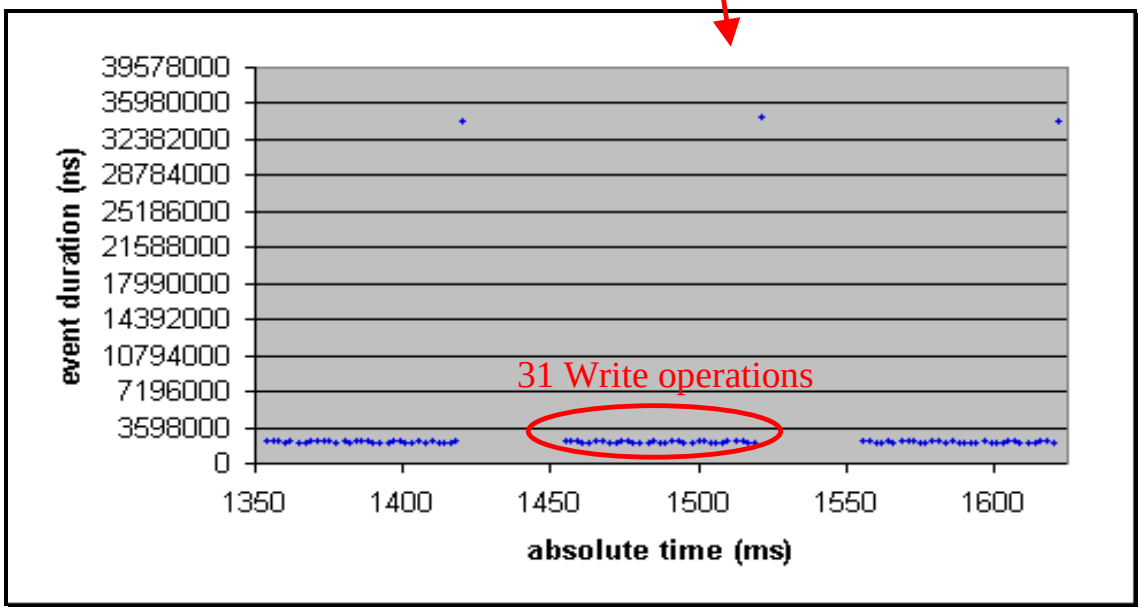

Figure 4.11-14 FS-d-1.a.1b – exploded view

4.11.8 Write (FS-d-1.a.10b)

- Number of threads: 1
- Memory model: a
- Number of bytes: 10 blocks (5120 bytes)

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
1000	2181510	59758020	14143261,9

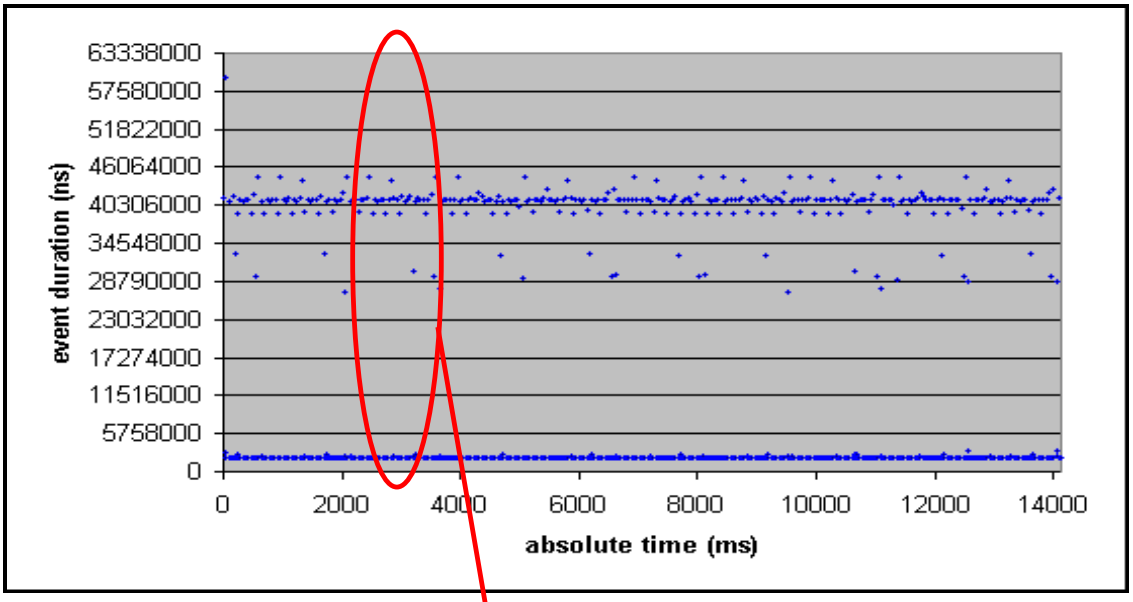

Figure 4.11-15 FS-d-1.a.10b

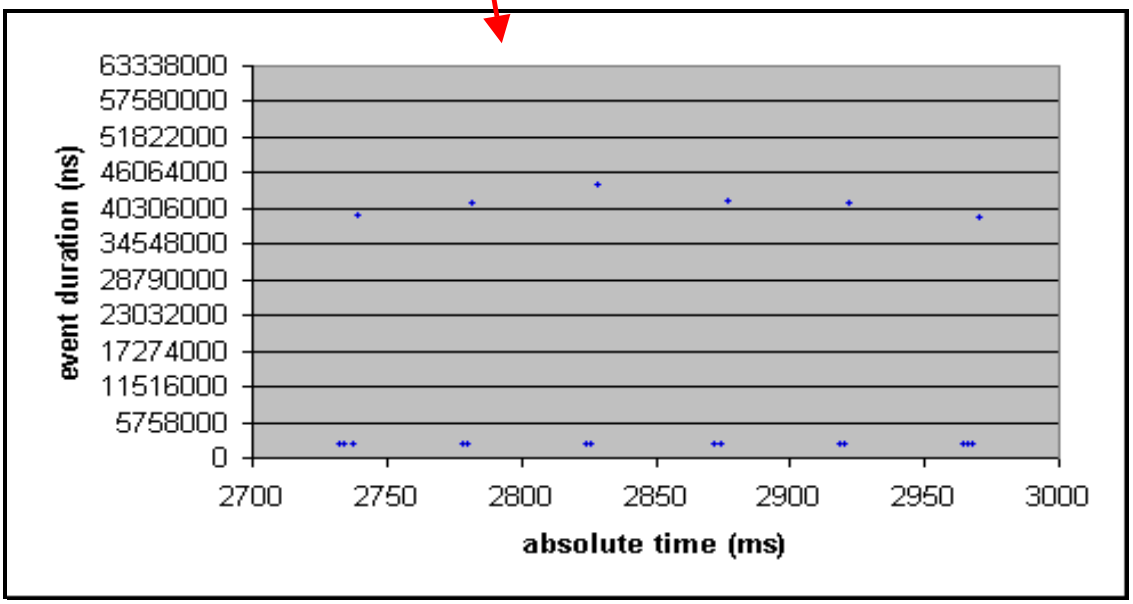

Figure 4.11-16 FS-d-1.a.10b – exploded view

4.12 Clock Interrupt Service Routine

In this section, the clock ISR is discussed. This topic is not included in the standard evaluation framework, but is included as an additional paragraph. VxWorks 5.3.1 is available for a wide variety of hardware platforms, which may have a different impact on the real-time behavior of the RTOS.

In this section, we make a distinction between two kinds of target-hardware architectures:

- Targets where the clock interrupt has the highest priority.
- Targets where the clock interrupt can be assigned any priority level.

Ordinary Intel based PCs (used for this evaluation) belong to the first category. The clock interrupt is assigned the highest priority, higher than all other external interrupts. This means that every action in the system (including the servicing of an important external hardware interrupt) can be interrupted to service the clock. It is therefore important to know the maximum time it can take to execute the clock ISR, as this latency needs to be added to other interrupt latencies to get the worst case result. This makes the system less responsive. Hardware architectures that have the clock interrupt at the highest priority are therefore a less common choice for the use in real-time systems.

The target-hardware architectures in the second category do not necessarily have the clock interrupt at the highest priority level. The system will therefore remain as responsive as possible to other external hardware interrupts (all the ones with a higher priority level than the clock), since their handling will not be delayed by the clock. One could argue that such architectures therefore make a more appropriate choice for the use in real-time systems. However, there is one drawback : the jitter on the clock caused by the low priority level of the clock interrupt. This may be unacceptable to certain types of applications where a precise clock tick frequency is necessary.

Three tests were executed in this section to study the execution time of the clock ISR. The main thread of each test is the same: a repetitive execution of a calculation that takes about 1 ms to execute (995.10 µs to be precise). Since our clock in VxWorks ticks every 16.7 ms, 1 out of every 17 calculations will be interrupted by the clock, resulting in a longer calculation time.

The first test, for which the results are displayed in section 4.12.1, consists of the main thread only. As can be seen in Figure 4.12-1 (p.77), an uninterrupted calculation takes 995 µs to execute. The second band of samples located at 999 µs are due to the clock interrupting the calculation. A conclusion that can be made is that a standard clock ISR takes about 4 µs to execute (on a Pentium 200 MHz). However, three occurrences are shown where the clock ISR took 19 µs to execute.

The second test (section 4.12.2) is basically the same as the first one, except that the system is loaded with 400 threads. The priority FIFO scheduling mechanism is used. This to verify whether or not the execution of the clock ISR depends on the number of threads queued. This does not seem to be the case. The average clock ISR took about 4 µs to execute, while again 3 occurrences of 20 µs or more were observed.

The third test (section 4.12.3) consists of the same main thread performing the calculations, plus one extra thread which installs a timer to expire cyclical every second. Note that the priority of the second thread is lower than that of the main thread to make sure neither the (second) thread or its signal handler executes and falsifies the measurements.

The results, displayed in Figure 4.12-5 (p. 79) are rather interesting. The mainstream clock interrupt still takes 4 µs to execute, but the expiration of the installed timer that occurs every second adds an extra 19µs resulting in a total execution time of 23 µs for the clock ISR. Additional peaks (indicated by red circles) can be found. A possible explanation for this is that the system installs timers too, for internal purposes. The highest peak is probably due to 2 timers expiring at the same time: a timer installed by the system and the timer installed by the test application. This peak adds another 19 µs to 23 µs, which is the execution time for the ISR when one timer expires.

The two major chores to handle for the clock ISR are updating the clock counter register and managing timer expirations. The counter register has to be updated every time the clock ticks and is the minimal amount of work that has to be performed by the clock ISR over and over again. This is the scenario when the clock ISR only takes 4 µs to execute.

On the other hand, when the clock ISR executes, updates the counter register and notices that a timer happens to expire, additional work needs to be performed, like sending the specified signal to the task that installed the timer handler. In our test, this additional work apparently took 19 µs. Since the maximum in Figure 4.12-5 (p. 79) added an extra 19 µs to the execution of the clock ISR, it is therefore reasonable to assume that 2 timers expired at the same time.

Finally, it is worth mentioning that the “execution time of the clock ISR” as described in this section includes three latencies :

- The interrupt latency (time needed to go from the interrupted task to the clock ISR).
- The work performed in the clock ISR (as described above).
- The interrupt dispatch latency (time needed to go from the clock ISR back to the interrupted task).

The reader will notice that the sum of the interrupt latency and interrupt dispatch latency, as measured in sections 4.2 and 4.3, is already bigger than 4 µs. This seems to be in contradiction with the observations made above that the average “execution time of the clock ISR” was only 4 µs.

However, this is due to the fact that we used a interrupt handler for our tests, which is a C function that is called from within the real ISR (because an interrupt cannot vector immediately to a C function). This extra function call results in a bigger interrupt (dispatch) latency. The clock interrupt on the other hand, was served by the system, immediately in the ISR. For a detailed discussion on interrupt handling, the reader is referred to section 3.2.2.3.

4.12.1 Clock ISR – no system load – no timers

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	995100	1014210	995333.4

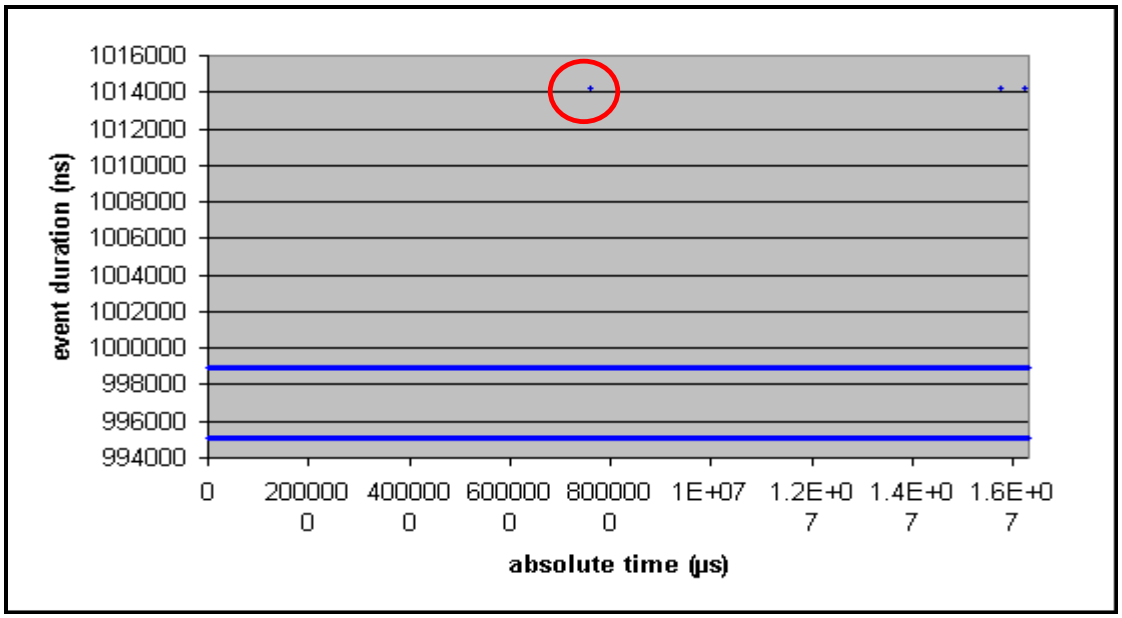

Figure 4.12-1 Clock ISR (no load, no timers)

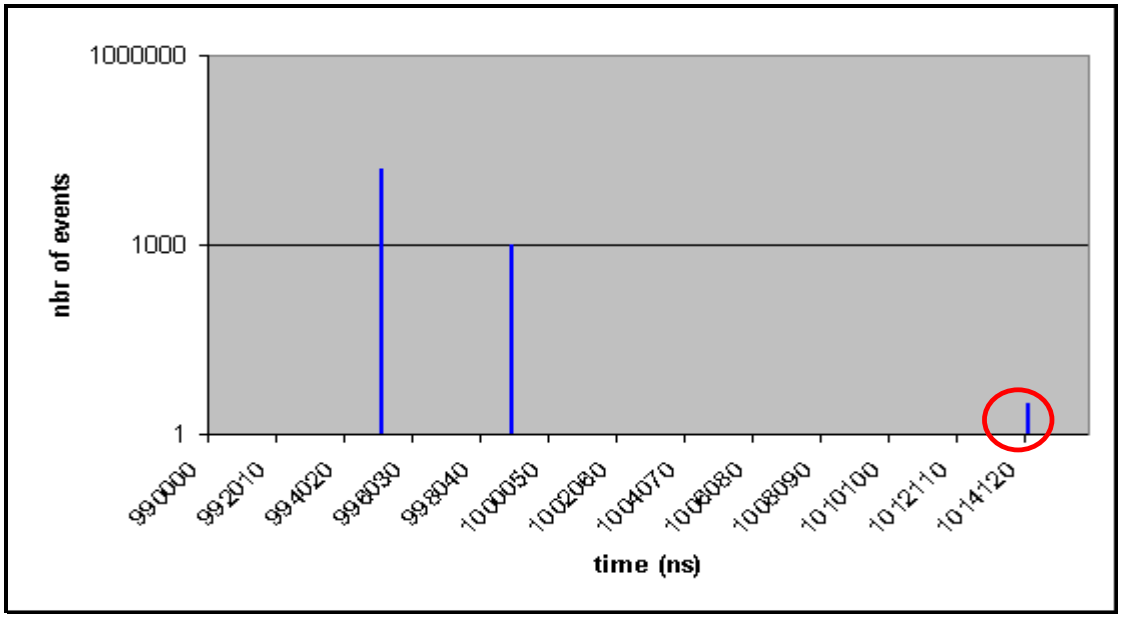

Figure 4.12-2 Clock ISR (no load, no timers) – frequency distribution

4.12.2 Clock ISR – with system load (no timers)

- Number of threads: 500
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16383	995100	1018020	995334.8

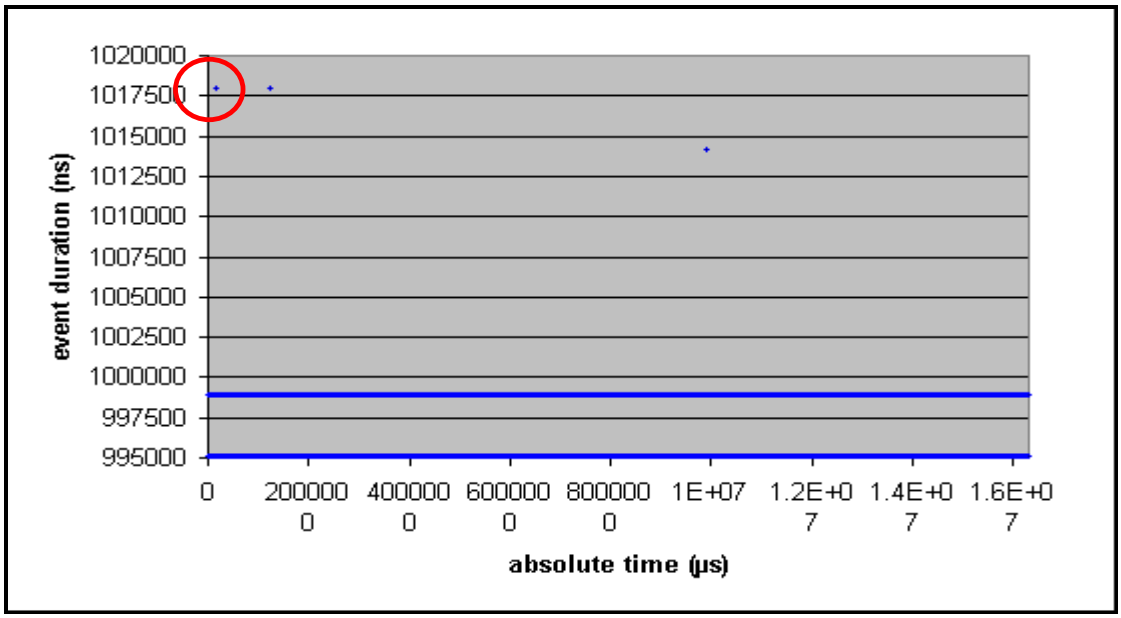

Figure 4.12-3 Clock ISR – with load (no timers)

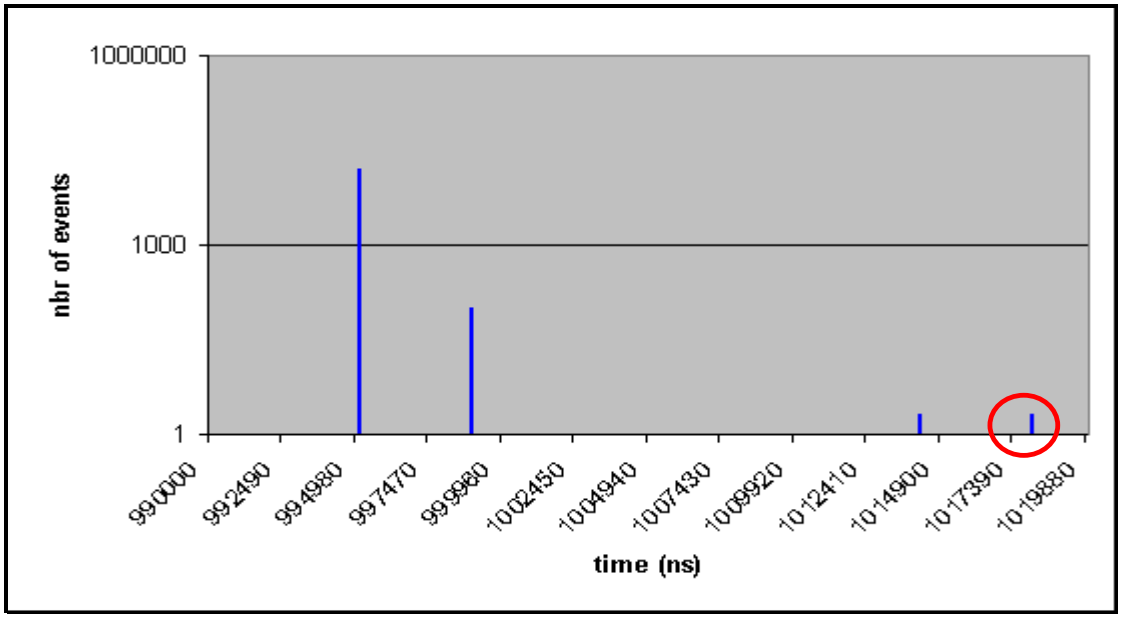

Figure 4.12-4 Clock ISR – with load (no timers) – frequency distribution

4.12.3 Clock ISR – with timers (no system load)

- Number of threads: 2
- Memory model: a

Number of samples	Minimum (ns)	Maximum (ns)	Average (ns)
16382	995100	1037220	995352.5

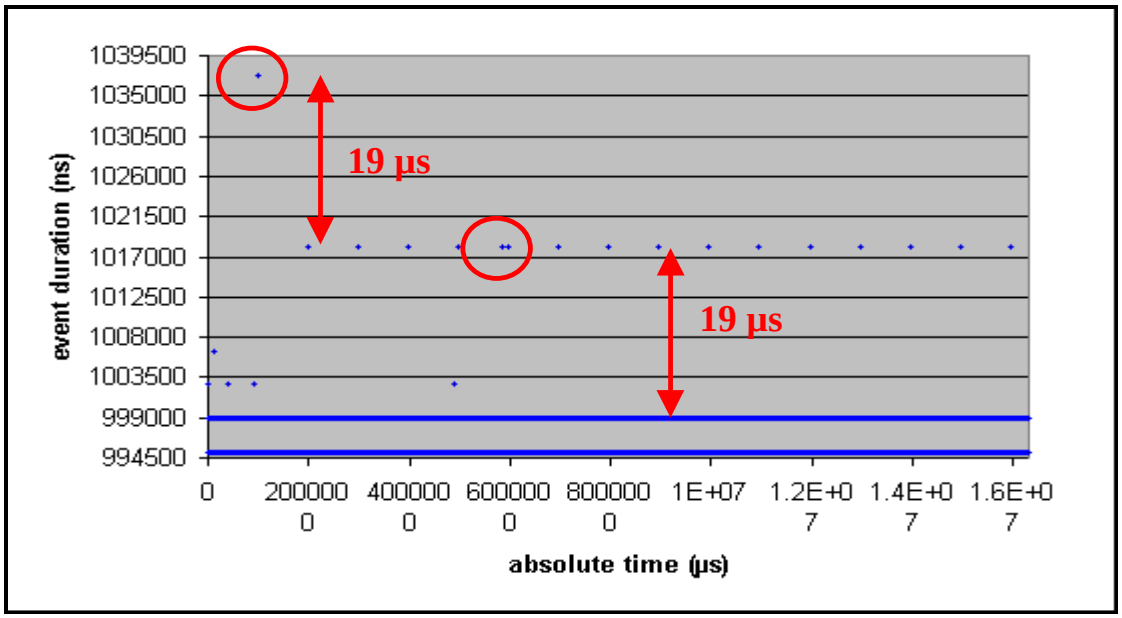

Figure 4.12-5 Clock ISR – with timers (no load)

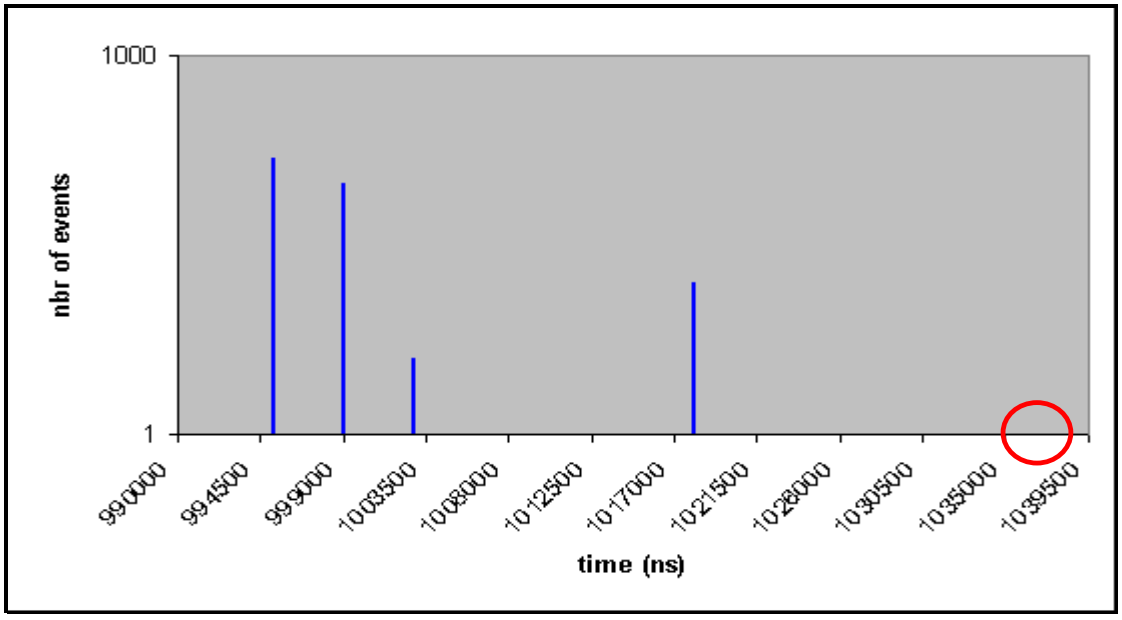

Figure 4.12-6 Clock ISR – with timers (no load) – frequency distribution

http://www.dedicated-systems.com
email: info@dedicated-systems.com

5 License policy and pricing

A single development user license has a list price of $16,500, or $29,500 for a five-pack. An annual maintenance contract is available for $3,000. This maintenance contract includes (among other things) access to 24 hours/day online technical support and toll free international phone access. For more information, please contact Wind River Systems.

A single run-time license runs $1,000. Discounts are available for volume purchases : $420 for 100 licenses, and $1 for 1 million run-time licenses.

6 Conclusions

😐 VxWorks 5.3.1 has a “client-server” architecture in the sense that it has a small microkernel which only handles the basic real-time features. All the other functionality is implemented as processes. VxWorks is not a message based operating system.

☺ During all of our tests, VxWorks 5.3.1 exhibited fast and predictable behavior.

We did find that the time needed to add a thread to a semaphore’s queue (and sort the queue) is proportional with the number of threads already in the queue. VxWorks could have used queue structures that result in a faster and more constant sorting times are available, but they require more memory, which might be a problem for targets with a limited memory capacity.

☺ VxWorks 5.3.1 has an extensive API and has a lot of additional libraries and device drivers for a wide variety of devices.

☺ Tornado 1.0.1 is a good and complete development environment with a very extensive set of tools. Furthermore, Tornado is an “open” environment, allowing for the integration of tools from third party vendors.

☺ Wind River Systems’ technical support team is very professional. Furthermore, users with a valid maintenance contract have access to 24 hours/day online technical support, consisting of a large and well-organized database containing thousands of articles.

☹ The manuals are not always clear. The documentation is not detailed enough on the system’s architecture and the description of some API function calls.

☹ The clock ISR execution time in VxWorks 5.3.1 can get pretty lengthy, especially when timers are used. We managed to create a situation where the clock ISR took more than 40 µs to execute. This slows the system down considerably, especially in platforms where the clock interrupt has the highest priority (like the x86 PC platform used for this evaluation).

☹ All the tasks in VxWorks by default share one common address space, without any memory protection. Memory protection is available, but an optional component (VxVMI) needs to be purchased and installed.

☺ VxWorks is available for a wide variety of target hardware platforms.

http://www.dedicated-systems.com
email: info@dedicated-systems.com

7 References

[1] VxWorks 5.3.1 reference manual – edition 1.
[2] VxWorks 5.3.1 Programmer's Guide – edition 1
[3] Tornado 1.0 User's Guide (Windows Version)
[4] VxWorks – Host and Target Machine © Wind River Systems, Inc
[5] VxWorks system architecture © Wind River Systems, Inc

8 Other publications

RTOS evaluation kick off!

Updated version from paper published in Real-Time Magazine, 98/3 - p. 3. Real-Time Consult started a research program about two years ago to qualify and compare real-time operating systems (RTOS). Now, a description and a framework for an evaluation project are defined and ready for use. This paper gives an overview and a framework for that project.

How to choose an RTOS

This paper explains how you should choose a RTOS correctly. It also explains what type of reports we make available and how you should use them.

What makes a good RTOS.

What is a real-time operating system (RTOS)? There are a lot of misconceptions on the topic of real-time. Following reflects the opinion of Real-time Consult on what makes a good RTOS. We will examine techniques that can be found in general purpose operating systems (GPOS) and explain why they can or cannot be used in real-time operating systems (RTOS).

Windows NT real-time extensions, better or worse ?

This article presents an overview of the capabilities of the real-time extensions for Windows NT 4.0 as a real-time operating system (RTOS). The extensions that are covered are RTX 4.1 from VenturCom, INTime 1.20 from Radisys and Hyperkernel 4.3 from Imagination Systems.

Can Windows NT 4.0 be used as an RTOS?

When Windows NT was originally designed it was not foreseen to be a real-time operating system. Yet many people are keen to use it as such and some already do so. Why would they want to do this and what are the advantages of Windows NT? This paper looks into those questions and explains the general architecture of the operating system. It also gives some results of performance tests that are useful for anyone who's interested in using Windows NT in real-time systems.

Is Windows CE 2.0 a real threat to the RTOS world ?

Paper published in Real-Time Magazine, 98/3 - p. 20. This article gives an overview of the capabilities of Windows CE 2.0 as a real-time operating system (RTOS).

Windows NT 4.0 workstation evaluation report.

This paper is a full evaluation report of Windows NT 4.0 workstation from Microsoft Inc.

RTX 4.2 real-time extension evaluation report.

This paper is a full evaluation report of RTX 4.2 real-time extension from VenturCom, Inc.

INtime 1.20 real-time extension evaluation report.

This paper is a full evaluation report of INtime 1.20 real-time extension from Radisys Corporation Ltd.

Hyperkernel 4.3 real-time extension evaluation report.

This paper is a full evaluation report of Hyperkernel 4.3 real-time extension from Imagination Systems,Inc

http://www.dedicated-systems.com
email: info@dedicated-systems.com

Comparison between Hyperkernel 4.3, RTX 4.2 and INtime 1.20

This paper is a summary of the evaluation reports for these three products and highlights the decision critical information.

RTX 4.2 real-time extension evaluation report – executive summary

The executive summary of the RTX 4.2 real-time extension evaluation report.

INtime 1.20 real-time extension evaluation report – executive summary

The executive summary of the INtime 1.20 real-time extension evaluation report.

Hyperkernel 4.3 real-time extension evaluation report – executive summary

The executive summary of the Hyperkernel 4.3 real-time extension evaluation report.

VxWorks/x86 5.3.1 evaluation report

This paper is the full evaluation report of the x86 based release of VxWorks 5.3.1 from WindRiver Systems Inc.

pSOSystem/x86 2.2.6 evaluation report

This paper is the full evaluation report of the x86 based release of pSOSystem 2.2.6 from Integrated Systems Inc.

QNX 4.25 evaluation report

This paper is the full evaluation report of QNX 4.25 from QNX Software Systems Ltd.

Comparison between VxWorks/x86 5.3.1, QNX 4.25 and pSOSystem/x86 2.2.6

This paper is a summary of the evaluation reports for these three products and highlights the decision critical information.

http://www.dedicated-systems.com
email: info@dedicated-systems.com

9 Abbreviations

	Explanation
API	Application Programming Interface
ISR	Interrupt Service Routine
IST	Interrupt Service Thread
MMU	Memory Management Unit
PIC	Programmable Interrupt Controller
FIFO	First In First Out
OS	Operating System
RTOS	Real-Time Operating System
GPOS	General Purpose Operating System