



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

QNX® NEUTRINO RTOS V6.2

© Copyright Dedicated Systems Experts. All rights reserved, no part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of Dedicated Systems Experts.

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



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

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.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

1	Summary	6
1.1	Product.....	6
1.2	Positive points.....	6
1.3	Negative points	6
1.4	Ratings.....	6
1.5	Pricing	6
2	Introduction	7
2.1	Different steps in creating such evaluation report	7
2.2	Revision strategy	7
3	Technical evaluation.....	8
3.1	Installation and Configuration	8
3.1.1	Installation	8
3.1.2	Configuration.....	8
3.2	RTOS Architecture.....	10
3.2.1	System Architecture	10
3.2.2	Basic System Facilities	11
3.3	API richness.....	13
3.3.1	Task Management	13
3.3.2	Clock and Timer	14
3.3.3	Memory Management	15
3.3.4	Interrupt Handling.....	16
3.3.5	Synchronization and Exclusion Objects.....	16
3.3.6	Communication and Message Passing Objects	19
3.4	Internet Support	21
3.5	Tools	22
3.6	Documentation and Support.....	25
3.7	Development methodology	26
4	Practical evaluation	27
4.1	System under test.....	27
4.1.1	Operating system and application configuration.....	27
4.1.2	Additional comments.....	27
4.1.3	Test results compared with QNX 6.1	27
4.2	Interrupt handling	28
4.2.1	One interrupt – no rescheduling.....	29
4.2.2	One interrupt – Rescheduling	32
4.2.3	Two simultaneous interrupts	34
4.2.4	Nested interrupt handling.....	37
4.2.5	Maximum sustainable interrupt frequency – ISR level.....	47
4.3	Threads.....	50
4.3.1	Creation (TF-a-1.d)	51
4.3.2	Deletion (TF-b-1.d).....	52
4.4	Thread switch latency – same process	53
4.4.1	Thread switch latency (TSL-a-2.d).....	54



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.4.2	Thread switch latency (TSL-a-10.d)	55
4.4.3	Thread switch latency (TSL-a-128.d)	56
4.5	Thread switch latency – different process	57
4.5.1	Thread switch latency (TSL-b-2.d)	58
4.5.2	Thread switch latency (TSL-b-10.d)	59
4.5.3	Thread switch latency (TSL-b-128.d)	60
4.6	Counting semaphores	61
4.6.1	Creation (SEO-a-1.d)	62
4.6.2	Deletion (SEO-b-1.d)	63
4.6.3	Deletion - after use (SEO-c-1.d)	64
4.6.4	No contention – Acquire (SEO-d-1.d)	65
4.6.5	No contention – Release (SEO-e-1.d)	66
4.6.6	Synchronization (SEO-f-63.d)	67
4.6.7	Acquiring a semaphore that is non-signaled	68
4.7	Mutexes	69
4.7.1	No contention – Acquire (SEO-d-1.d)	70
4.7.2	No contention – Release (SEO-e-1.d)	71
4.7.3	Priority inversion (SEO-g-3.d)	72
4.8	File system – Proprietary	73
4.8.1	Creation	74
4.8.2	Deletion	75
4.8.3	Read – 1 byte	76
	Read – 512 bytes	77
4.8.4	Read – 5120 bytes	78
4.8.5	Write – 1 byte	79
4.8.6	Write – 512 bytes	80
4.8.7	Write – 5120 bytes	81
4.9	Network stack – TCP/IP	82
4.9.1	Full TCP/IP Stack – Receive Capacity	83
4.9.2	Tiny TCP/IP Stack – Receive Capacity	84
4.9.3	Full TCP/IP Stack – Send Capacity	85
4.9.4	Tiny TCP/IP Stack – Send Capacity	86
4.10	Clock Interrupt Service Routine	87
4.10.1	Clock ISR – no system load – no timers	89
4.11	Maximum number of objects	90
4.11.1	Semaphores	91
4.11.2	Threads	92
4.12	Memory Leaks	93
4.12.1	Semaphore and Mutex Creation/Deletion	94
4.12.2	Process Creation/Deletion	95
5	License policy and pricing	100
6	Conclusions	101
6.1	Comparison with QNX 6.1 (previous version)	101
6.1.1	Ratings	101



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

6.1.2 Test Results	101
7 Appendix A: Commentary from the vendor	103
8 Appendix B: Abbreviations	104
9 Appendix C: Document revision history	105
9.1 Issue 2.50 (August 13, 2002).....	105

1 Summary

1.1 Product

The QNX NEUTRINO RTOS v6.2.0, from QNX Software Systems Ltd.

1.2 Positive points








- Fast performance
- Excellent architecture for a distributed and robust system
- Good platform support

1.3 Negative points

- Slow Integrated Development Environment

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 our website (<http://www.dedicated-systems.com/encyc>)

Installation and Configuration	0		10
RTOS Architecture	0		10
API Richness	0		10
Internet support	0		10
Tools	0		10
Documentation and Support	0		10
Test Results	0		10

1.5 Pricing

Please contact the vendor for detailed and up-to-date pricing information.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

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 Dedicated Systems Experts 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 Dedicated Systems Experts. 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. Comments which we feel are justified are incorporated into the report.

2.2 Revision strategy

To improve the quality of these reports, the reader is encouraged to send us by email (info@dedicated-systems.com) 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 our website (<http://www.dedicated-systems.com>). This document defines (among other things) the different memory protection models and 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

								8		
--	--	--	--	--	--	--	--	---	--	--

 10

Installation is quick and simple. Creating and configuring a custom QNX image is done in the IDE or by text-based build files.

3.1.1 Installation

The QNX NEUTRINO RTOS v6.2 is quick and easy to install. The easiest way is to install the QNX NEUTRINO RTOS v6.2 from a CD, which can be ordered from QNX Software Systems. After only a few minutes the basic modules are installed i.e., the kernel and a user interface (Photon microGUI windowing system). Additional packages like compilers, etc., can be installed by means of the package manager.

3.1.2 Configuration

Configuring the QNX NEUTRINO RTOS v6.2 is reasonably straightforward. When installing the full environment, the most critical components like storage devices and network cards are detected automatically. If further configuration is necessary, it can be done through the graphical user interface.

Building a custom QNX image is done through build files. Modules can be added, removed and configured by manually editing these text-based files. The documentation contains plenty of examples of such build files.

In addition, the IDE includes a system builder tool, which enables the management of QNX images. It replaces the build files with a graphical tool to create images (both boot images and flash images) and allows the importing of existing build files. The system builder tool features dependency analysis (tells you which libraries might be missing), as well as a “dietician”, which creates smaller versions of the shared libraries you’re using that only contain the functions you need.

Although launching the Momentics IDE takes a huge amount of time (some minutes), the System builder tool works reasonable fast and is pretty easy to use. In the beginning you will need to experiment a bit to understand where some modules are located (for instance specific devices are included in DLLs). This could be improved by adding a comment column in the selection process. Once a module is included in the system builder a comment section shows up in the IDE explaining the usage and options of the library: very good! The System Builder will also automatically add the shared libraries needed by the added module in your target system configuration.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

The System Builder is best initialized with an existing (simple) build file, which sets then already the basic configuration in a good shape.

The “diet” function works very well and is something we haven’t found on any other graphical platform builder yet. Of course to “diet” shared libraries is a slow process (checking all dependencies), but this is something that needs only to be done at the end of the development cycle.

The graphical System Builder Tool, with its dependencies checker and diet function, are a major improvement compared with the QNX v6.1.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

3.2 RTOS Architecture

RTOS Architecture

0 9 10

True client-server architecture, providing full virtual memory protection. The QNX NEUTRINO RTOS v6.2 is a message based OS, and can seamlessly be distributed over multiple nodes. The RTOS supports SMP, and implements several HA (High Availability) features.

3.2.1 System Architecture

☺ The QNX NEUTRINO RTOS v6.2 has a client-server architecture consisting of a microkernel and optional cooperating processes. The microkernel implements only the core services, like threads, signals, message passing, synchronization, scheduling and timer services. The microkernel itself is never scheduled. Its code is executed only as the result of a kernel call, the occurrence of a hardware interrupt or a processor exception.

Additional functionality is implemented in cooperative processes, which act as server processes and respond to the request of client processes (e.g. an application process). Examples of such server processes are the file system manager, process manager, device manager, network manager, etc. While the kernel runs at privilege level 0 of the Intel processor, the managers and device drivers run at levels 1 and 2 (to perform IO operations). Application processes on the other hand run at privilege level 3, and can therefore only execute general instructions of the processor.

☺ The QNX NEUTRINO RTOS v6.2 is a message-based operating system. Message passing is the fundamental means of inter-process communication (IPC) in this RTOS. The message passing service is based on the client-server model: the client (e.g. an application process) sends a message to a server (e.g. device manager) who replies with the result. A lot of the QNX NEUTRINO RTOS API calls use the message passing mechanism. For example, when an application process wants to open a file, the system call is translated into a message that is sent to the file system manager. The file manager (after accessing the disk via its device drivers) replies with a file handle. This message passing mechanism is network transparent i.e., the system can be seamlessly distributed over several nodes, without requiring any changes in the application code.

When passing messages between client and server, the QNX NEUTRINO RTOS v6.2 uses a mechanism called "client-driven priority". This means that a server process inherits the priority level of the client process requesting a service. When the client's request is serviced, the server process can regain its original priority level. When multiple clients are requesting a service from a server, the server process assumes the priority level of the highest priority client process. This is to avoid priority inversion.

A client-server architecture has many advantages, one of which is robustness. Every manager (except for the process manager) and device driver runs in its own virtual memory address space, resulting in a robust and reliable system. This price to pay is performance: execution of system calls results in a few context switches (with an overhead caused by memory protection) resulting in somewhat lower performance.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

3.2.2 Basic System Facilities

3.2.2.1 Task Handling Method

The QNX NEUTRINO RTOS v6.2 uses processes and threads. A process defines the address space in which threads will run, and will always contain at least one thread.

☺ There are 63 distinct thread priority levels available to applications. This is acceptable, but it is sometimes desirable to have 128 levels. This is especially so for large real-time applications that are being designed using techniques like RMA (Rate Monotonic Analysis), where every thread needs to have its own distinct priority level.

The scheduler schedules the threads according to a prioritized FIFO, round-robin algorithm and something new for the QNX 6.2 release: “sporadic scheduling”.

Sporadic is a sort of adaptive scheduling: e.g. the thread priority is decreased if it consumes too much CPU during a time-slice. After some time on a low priority level (replenishment period) the thread can go back to its original priority. All these parameters (priority boundaries included) are adaptable for each thread.

QNX 6.2	
Model	Threads and processes
Priority levels	64
Max. number of tasks	4095 processes Every process can have 32767 threads
Scheduling policies	Prioritized FIFO Round-robin scheduling Adaptive ¹ Sporadic ²
Number of documented states	14

3.2.2.2 Memory Management Method

☺ In the QNX NEUTRINO RTOS v6.2, every process has its own virtual memory space. The virtual memory is supported by the paging mechanism of the processor. Virtual memory protects processes (both user and system processes) from each other, which enhances the overall robustness of the system.

Swapping to disk is supported but needs to be explicitly enabled by means of the family of “unlock” calls. QNX Software discourages the use of these calls because they were not designed for general purpose,

¹ Only useful in a non-RT environment.

² This is a new feature in the QNX 6.2 system.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

but simply for accommodating the GNU compiler and linker. Swapping is usually undesirable in embedded systems, but is becoming more and more necessary for host systems. This is why QNX plans to support transparent swapping in future releases.

Demand paging is the ability for the OS to load parts of an executable into memory as required. In the QNX NEUTRINO RTOS v6.2, the entire code segment of the binary is loaded into memory upon execution. Demand paging affects real-time predictability; hence it is not implemented in the QNX RTOS.

QNX 6.2	
MMU support	Yes
Physical page size	4K
Swapping/Demand Paging	Yes/No
Virtual memory	Yes
Memory protection models	Full virtual memory protection

3.2.2.3 Interrupt Handling Method

The interrupt redirector in the microkernel handles interrupts in their initial stage. This redirector saves the context of the currently running thread and sets the processor context such that the ISR has access to the code and data that are part of the thread the ISR is contained within (i.e., this is the thread that attached the ISR).

The QNX NEUTRINO RTOS v6.2 supports interrupt sharing. When an interrupt occurs, every interrupt handler attached to the hardware interrupt is executed in turn. The user should make no assumptions about the order in which interrupt handlers sharing an interrupt are invoked.

☺ The interrupts aren't disabled during the execution of an interrupt handler, so interrupts can be nested. Unmasked interrupts can be serviced during the execution of running interrupt handlers.

☺ Since the QNX NEUTRINO RTOS v6.2 is a message based client-server operating system, ISR to thread communication is based on signals and pulses. An ISR can send pulses and signals to threads and processes and can ready a thread which is waiting for the interrupt via the "InterruptWait" call. ISRs can also pass data through application-defined structures to the process that attached it. The use of semaphores in the ISR was intentionally blocked, because it is a less elegant solution for an RTOS with a message based architecture.

QNX 6.2	
Handling	Nested, prioritized
Context	The ISR runs in the context of the thread that attached it
Stack	The ISR has its own stack
Interrupt-to-task communication	Signals and pulses

3.3 API richness



The QNX NEUTRINO RTOS v6.2 provides both a POSIX-compliant and proprietary API. The API is geared towards message-based systems, which is a natural match for the system architecture.

The tables below make an inventory of basic real-time objects and features present in the POSIX and QNX proprietary interfaces. While interpreting these results, the reader should keep in mind that these tables cover a strictly defined set of system calls only. The QNX NEUTRINO RTOS v6.2 provides other interfaces that are not mentioned in the tables below.

3.3.1 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	✓
Disable scheduling	-
Total	14

Thread management	YES
Total in percentage	82%

3.3.2 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	7
Total in percentage	100%

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%

3.3.3 Memory Management

Fixed block size partition	NO
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	-
Total	0
Total in percentage	0%

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	✓
Get remaining free bytes	-

Non-fixed block size pool	YES
Lock/unlock pool in memory	✓
Lock/unlock block in memory	✓
Total	7
Total in percentage	54%

3.3.4 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	7
Total in percentage	88%

3.3.5 Synchronization and Exclusion Objects

Aside from the mechanisms mentioned below, the QNX NEUTRINO RTOS v6.2 provides additional means for *synchronizing* execution, some of which are very powerful (joined threads, reader/writer locks, sleepers, spinlocks, barriers, etc).

The QNX NEUTRINO RTOS v6.2 also supplies atomic_xxx functions which can be used for implementing high-speed operations without a need for traditional exclusion primitives.

Counting semaphore	YES
Get maximum count	-
Set maximum count	-
Set initial value	✓
Share between processes	✓

Counting semaphore	YES
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Post	✓
Post – Broadcast	-
Get status (value)	✓
Total	7
Total in percentage	70%

Binary semaphore	NO
Set initial value	-
Share between processes	-
Wait – blocking	-
Wait - non blocking	-
Wait - with timeout	-
Post	-
Get status	-
Total	0
Total in percentage	0%

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	✓

RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

Mutex	YES
Release	✓
Get status	-
Get owner's thread ID	-
Get blocked thread ID	-
Total	8
Total in percentage	67%

Conditional variable	YES
Pend non blocking	✓
Pend with timeout	✓
Pend in fifo / priority order	-
Broadcast	✓
Priority inversion	-
Total	3
Total in percentage	60%

Event flags	NO ³
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%

³ QNX 6.2 does not have event flags, but has several other mechanisms that provide the same – and even more – functionality.

POSIX signals	YES ⁴
Install signal handler	✓
Detach signal handler	✓
Mask/unmask signals	✓
Identify sender	✓
Set destination ID	✓
Set signal ID	✓
Get signal ID	✓
Signal thread	✓
Queued signals	✓
Total	9
Total in percentage	100%

3.3.6 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	✓
Send – OOB (out of band)	-
Send - with timeout	✓

⁴ On top of these signals, QNX supports all of the enhanced POSIX real-time signals (queued signals, signals with priority characteristics, etc).

Queue	YES
Send – broadcast	-
Timestamp	-
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%



RTOS EVALUATION PROGRAM

Doc. Name: QNX® Neutrino RTOS v6.2

Doc. Version: 2.50 Doc. date: 13 August, 2002

3.4 Internet Support

Internet support 0

								8		
--	--	--	--	--	--	--	--	---	--	--

 10

The QNX Momentics development suite contains the following products and tools:

- An Embedded Web Server. The embedded web server supports CGI 1.1, HTTP 1.1, and dynamic HTML (via Server Side Include commands).
You can also handle SSI by using a data server. The data server allows multiple threads to share data: having a process updating the data server about the state of a hardware device while the embedded web server accesses that state in a decoupled but reliable manner.
- A small-footprint Web Browser for viewing information. It has full HTML 3.2 support, frames support, javascript, cookies, etc. Even MPEG audio/video streaming is possible on the voyager browser. The Mozilla and Opera browsers have recently also been ported to run on the QNX NEUTRINO RTOS v6.2.
- Source is available to build internet-enabled applications into an embedded system.
- Broad networking and protocol support. The reader is referred to the QNX website for detailed information.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

3.5 Tools

Tools 0 8 10

There are different tools available for the QNX NEUTRINO RTOS v6.2. New for QNX Neutrino 6.2 is the QNX Momentics IDE based on the (extensible) eclipse Framework.

In addition, the Metrowerks Codewarrior IDE and the GCC toolkit can be used. Tools for both self-hosted and cross development are available. These toolkits contain the most commonly used tools.

The new IDE is an enhancement compared with the previous version of QNX. The IDE is delivered with good tutorial videos (on CD-ROM) giving you a quick start. Also, the user interface is intuitive.

The main disadvantage of the IDE is its huge consumption of resources (CPU and memory), so it's almost useless on older hardware platforms. Surely the time it takes to launch the IDE (about 5 minutes on our host which is an old fashion Pentium 200 MHz MMX with 128MB RAM) is unacceptable. Once the IDE is started, the speed is slow but workable: just take care not to close the IDE window accidentally.... This is a phenomena we detect more and more on new products: it seems that programmers are no longer used to working in a limited environment, and start to code their programs in a very inefficient way!

We also noticed some problems in redirecting compiler errors to the correct code line in the editor. Reinstalling the IDE resolved this problem.

Aside from the tools listed in the table below, additional tools are available.

- Photon Developer's toolkit to develop Photon user interfaces.
- TCP/IP Developer's toolkit to develop programs for peer-to-peer communication over TCP/IP connections.

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Editor					
Color highlight	Yes	✓			✓
Integrated help	Yes	✓			✓
Automatic code layout	Yes	✓			✓
Compiler					
C	Yes	✓	✓	✓	✓
C++	Yes	✓	✓	✓	✓

RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Java	Yes	✓	✓	✓	✓
Ada	No				
Assembler	Yes	✓	✓	✓	✓
Linker					
Incremental	Yes	✓	✓	✓	✓
Symbol table generation	Yes	✓	✓	✓	✓
Development tools					
Profiler	Yes	✓	✓	✓	✓
Project management	Yes	✓			✓
Source code control ⁵	Yes	✓	✓	✓	✓
Revision control	Yes	✓	✓	✓	✓
Debugger					
Symbolic debugger	Yes	✓	✓	✓	✓
Thread sensitive debugging	Yes	✓	✓	✓	✓
Mixed source and disassembly	Yes	✓			✓
Variable inspect	Yes	✓	✓	✓	✓
Structure inspect	Yes	✓	✓	✓	✓
Memory inspect ⁶	Yes	✓	✓	✓	✓
Register inspect	Yes	✓	✓	✓	✓
System analysis tool					

⁵ The Momentics IDE supports CVS and third-party products via plugins (for instance Clearcase). With or without a coupled source code control system, the Momentics Source Editor has an automatic history function where previous saved versions of your source files are stored. At any moment you can compare your file with some previous one (using a graphical tool) and choose the changes to revert to. The number of days your history is kept can be adapted.

⁶ Also a debug malloc tool can be used to run your application against an instrumented version of the malloc library, which allows you to trace the memory allocations as well as detect common memory-related errors.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Tracing	Yes	✓	✓	✓	✓
Thread information	Yes	✓	✓	✓	✓
Interrupt information	Yes	✓	✓	✓	✓
Loader					
TFTP boot loader	Yes	✓	✓	✓	✓
Serial boot loader	Yes	✓	✓	✓	✓
Load separate modules	No				



RTOS EVALUATION PROGRAM

Doc. Name: QNX® Neutrino RTOS v6.2

Doc. Version: 2.50 Doc. date: 13 August, 2002

3.6 Documentation and Support

Documentation and Support 0 10

We enjoyed excellent technical support during this evaluation, the documentation is improved a lot compared with the previous version of QNX (6.1).

☺ Clearly, QNX Software did read our report on their QNX 6.1 RTOS. We were happy to notice that they tackled our main criticism: the lack of documentation on API parameters. Now these are always described in the documentation. It could even be improved further if a link was provided to the used structures (if any) of these parameters.

Just like before, the documentation does a decent job giving a general overview of the system and its architecture.

QNX Software offers paid enhanced support options to its customers (Standard and Priority support).



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

3.7 Development methodology

Past versions of the QNX RTOS used the host = target approach only i.e., host and target are the same machine. With the QNX Neutrino RTOS v6.2, developers can choose from self-hosted and cross development.

As described in section 3.2, the QNX Neutrino RTOS v6.2 can be configured with only a microkernel, as well as with many other modules, turning it into a fully fledged multi-user operating system capable of serving as a development environment.

The advantage of the self-hosted approach is that the user has the option to do it all on one machine: the application can be tested on the same machine as it was developed on, debugging can be done locally, etc. There are no problems with communication between host and target.

Developers that prefer a standard MS-Windows or Solaris desktop to the QNX desktop can use cross-development tools. In addition to the QNX Momentics IDE, the Metrowerks IDE for MS-Windows can be used for compiling and debugging from the Windows based host machine.

There's often a lot of discussion about which development method is the better one: self-hosted or cross development. It all really depends on the quality of the tools. If the quality of the cross-development tools is poor, it is better to opt for self-hosted development, and vice versa. Unfortunately, evaluating the quality of the development tools is not within the scope of this report. Nonetheless, it is important advantage that an OS support both methods.




RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4 Practical evaluation

Test Results 0  10

The system is fast and predictable. We found no issues concerning performance, predictability or robustness during the tests.

4.1 System under test

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
- VMETRO PCI exerciser in PCI slot 3 (PCI interrupt level D, local bus interrupt level 10)
- VMETRO PBT-315 PCI analyzer in PCI slot 4.
- External and CPU internal cache was enabled during the tests, unless otherwise specified.
- DRAM timing was set to 60ns.

4.1.1 Operating system and application configuration

All tests were executed with a minimal kernel image, except for the file system and network stack tests. All test applications use the POSIX-compliant API wherever possible.

4.1.2 Additional comments

With reference to our document titled "Report definition and test plan", all tests are executed using memory model "d". This memory model features full virtual memory protection for all processes.

4.1.3 Test results compared with QNX 6.1

It's a good exercise to compare the test results with the previous version of the RTOS.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2 Interrupt handling

The following sections will test the system's ability to handle interrupts. The different test scenarios include:

- Single interrupts that do not cause rescheduling in the application.
- Single interrupts that cause rescheduling in the application.
- Two interrupts occurring simultaneously.
- Nested interrupts.
- Maximum interrupt frequency that can be sustained without losing interrupts.

All measurements were taken on ISR level. The QNX NEUTRINO RTOS v6.2 also provides a way to immediately service interrupts on thread level (without explicitly hooking an ISR), but this method was not used here.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.1 One interrupt – no rescheduling

For this test we created a high priority thread that installs an interrupt handler and then starts to continuously write traces to the PCI bus. Interrupts are being generated roughly every 50µs, 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 is finished executing, the thread that was interrupted resumes. 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 the interrupted task to the ISR, while the *interrupt dispatch latency* is the time needed to get back from the ISR to the task.

The reader should bear in mind that on a PC-based target, the clock interrupt has the highest priority by default (higher than all other external interrupts). Under these circumstances, the time needed to service this clock interrupt should be added to the interrupt latency to get the worst-case scenario results. For a discussion on the execution time of the clock ISR, the reader is referred to section 4.10.

4.2.1.1 Interrupt latency

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
155	1.6	4.3	1.7

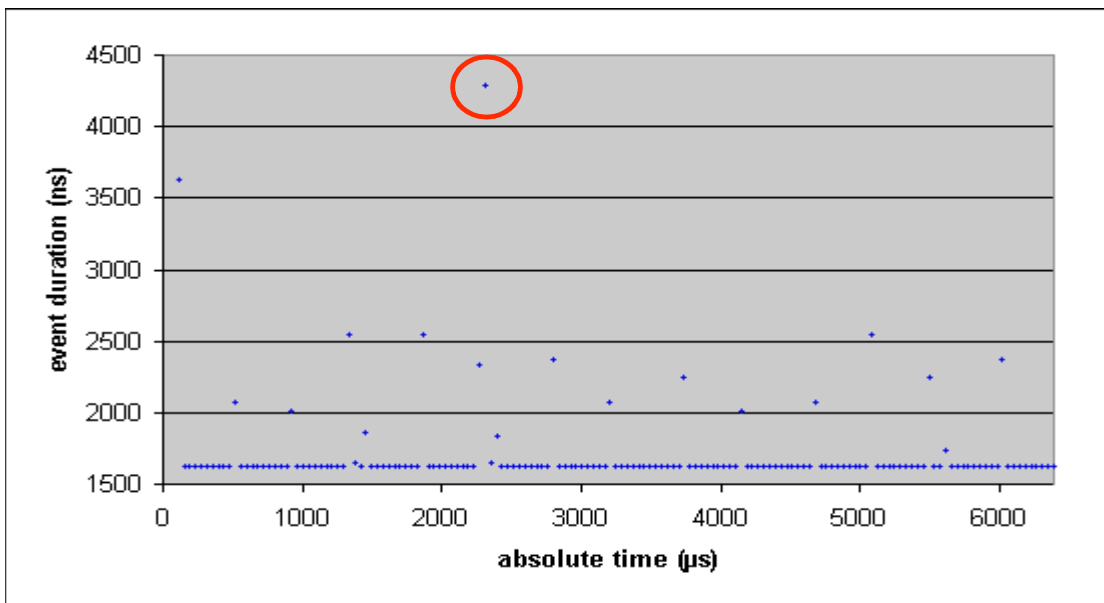


Figure 4.2-1 IL-a-1.d

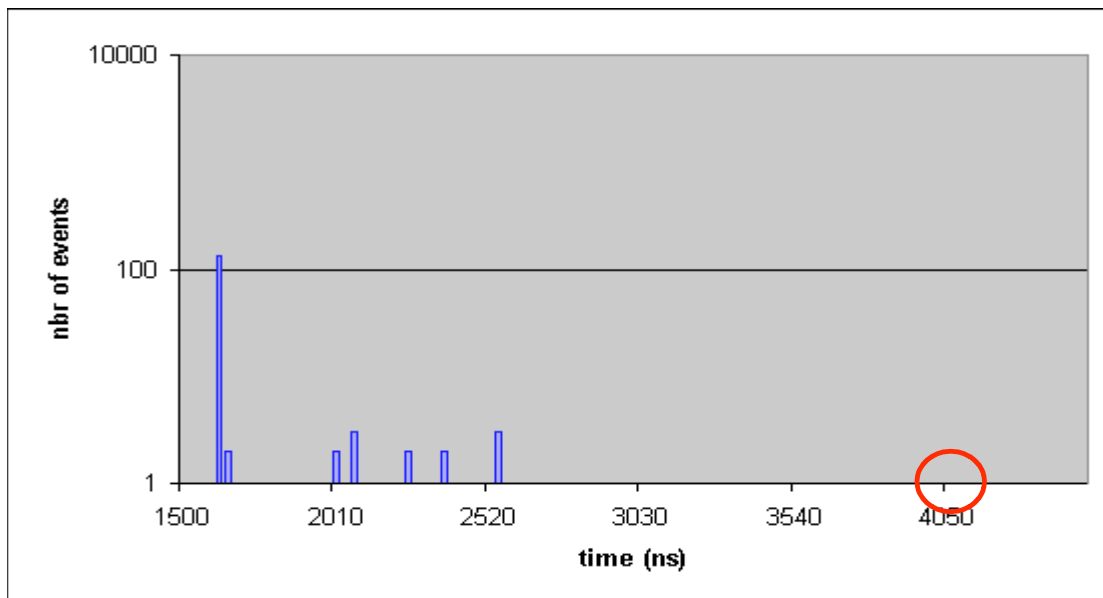


Figure 4.2-2 IL-a-1.d - frequency distribution

4.2.1.2 Interrupt dispatch latency

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
155	1.9	2.7	1.9

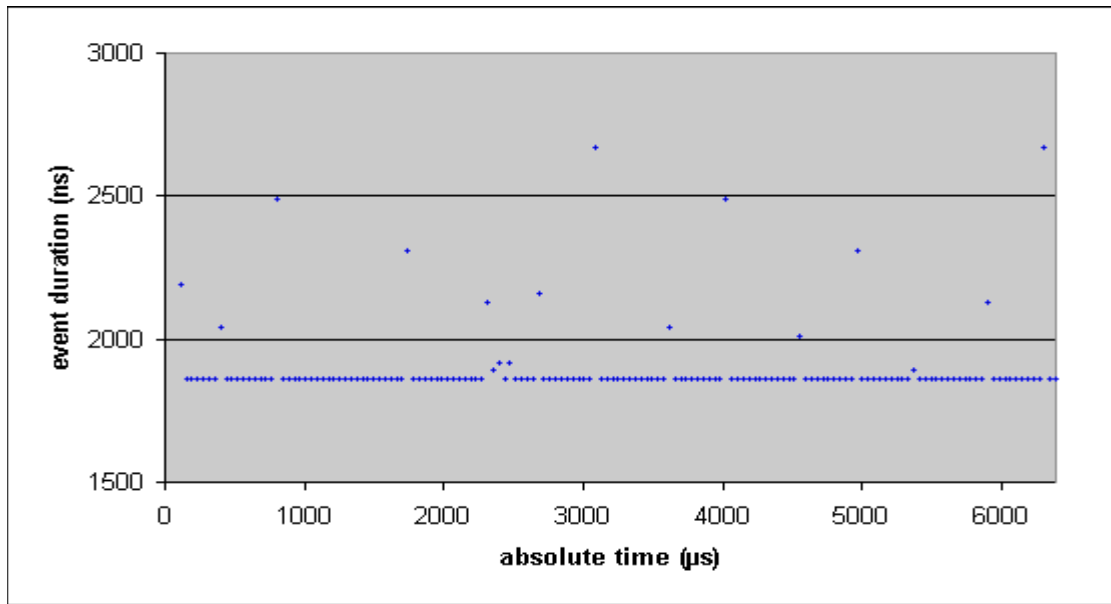


Figure 4.2-3 IDL-a-1.d

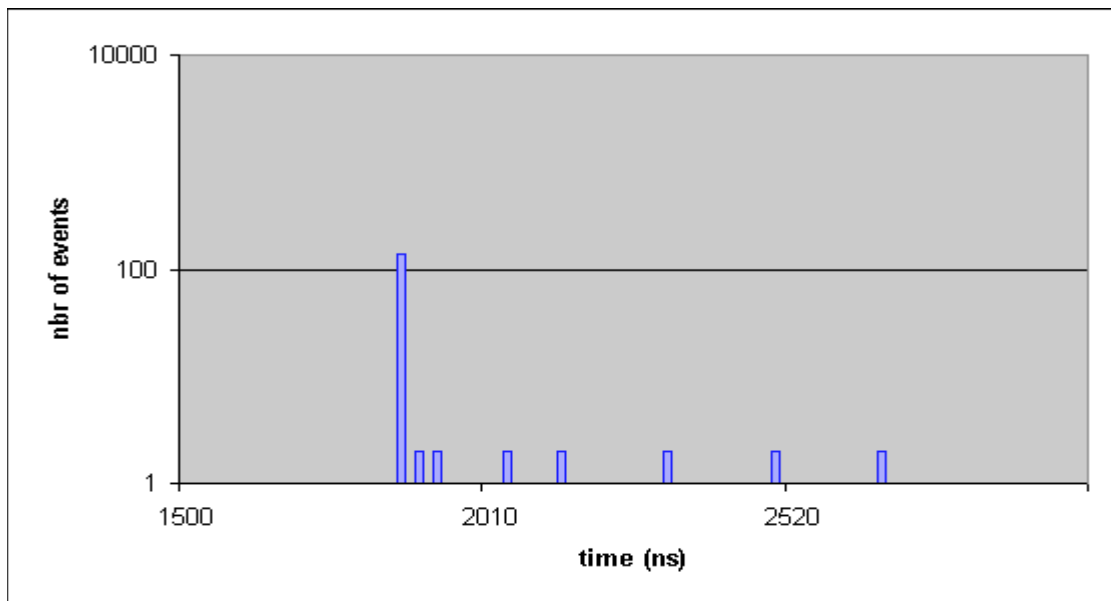


Figure 4.2-4 IDL-a-1.d - frequency distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.2 One interrupt – Rescheduling

The idea of this test is that the ISR executes a piece of code that releases a higher priority thread than the one that was interrupted by the ISR; hence the system needs to reschedule.

The interrupt dispatch latency measured in section 4.2.2.1 is the time the system needs to switch from the ISR to the thread that was signaled by the ISR.

4.2.2.1 Interrupt dispatch latency

- Number of threads: 2
- Memory model: d

Number of samples	Minimum (μs)	Maximum (μs)	Average (μs)
16383	2.3	7.7	2.3

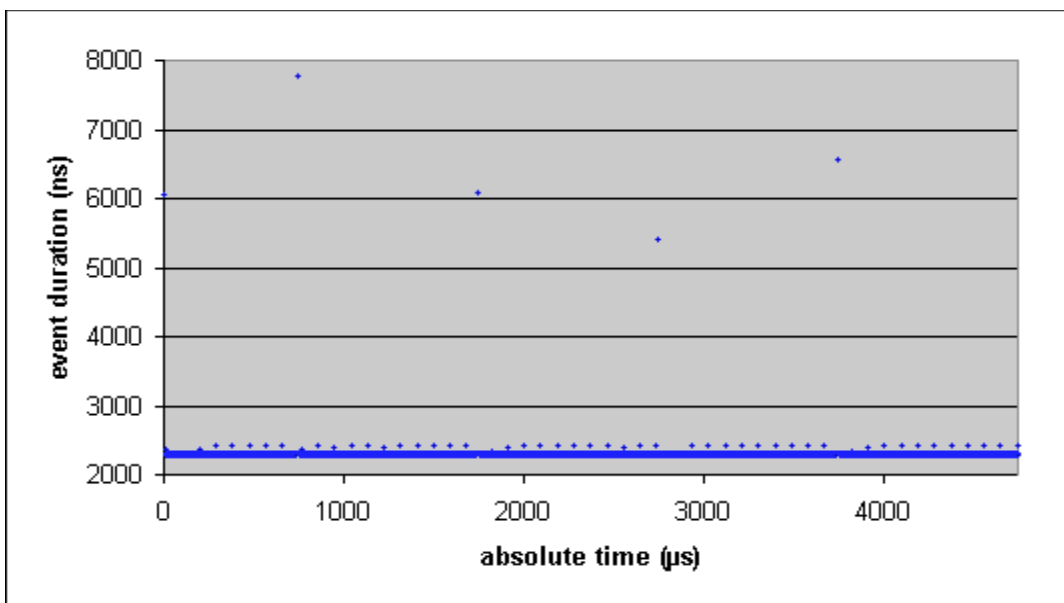


Figure 4.2-5 IDL-b-1.d

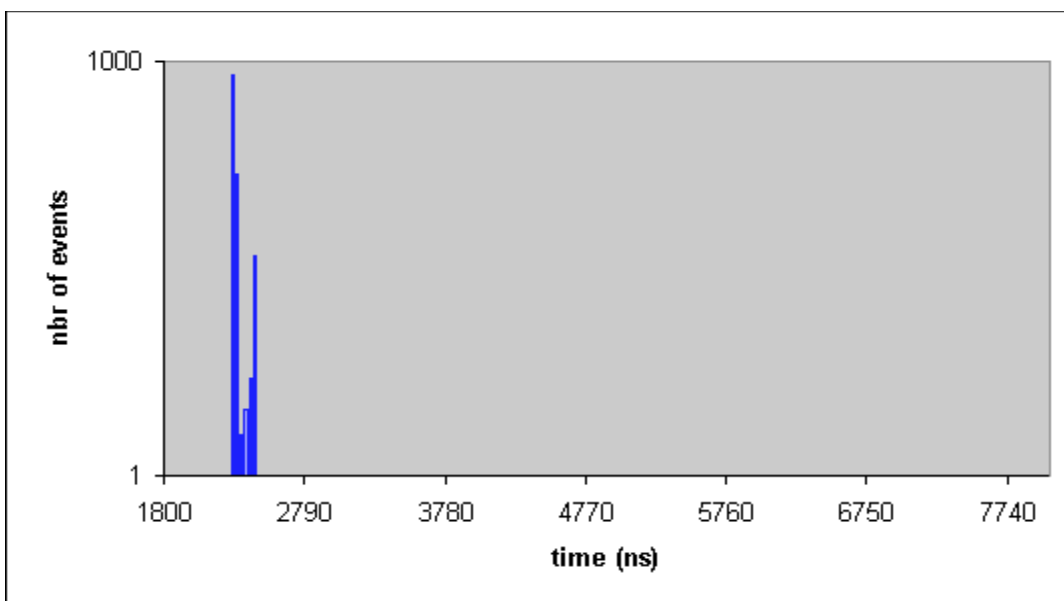


Figure 4.2-6 IDL-b-1.d - frequency distribution

4.2.3 Two simultaneous interrupts

In this test, two (nearly) simultaneous interrupts are generated. This is accomplished by adding a second PCI exerciser (PDrive) to the system.

- PDrive_HI (generates the high priority interrupt) uses PCI interrupt line C (INTC#) and generates interrupts with IRQ level 9.
- PDrive_LO (generates the low priority interrupt) uses PCI interrupt line B (INTB#) and generates interrupts with IRQ level 10.

Two interrupt service routines (ISR_HI and ISR_LO) are attached to IRQ 9 and IRQ 10 respectively. The interrupt latency was measured for both ISR_HI and ISR_LO. The interrupt latency was measured as the time elapsed between the last instruction of the interrupted thread and the first instruction of the ISR.

The time difference between the two generated interrupts was smaller than 100ns, which definitely qualifies as simultaneous interrupts.

☺ The system does not have any problems dealing with these interrupts. The high priority interrupt was serviced first by the system, with a maximum interrupt latency of 3µs. The low priority interrupt is serviced as soon as the high-priority interrupt is completely handles, but was still serviced less than 6µs after both interrupts occur.

4.2.3.1 Interrupt latency – ISR_HI

- Number of threads: 2
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	1.6	2.5	1.6

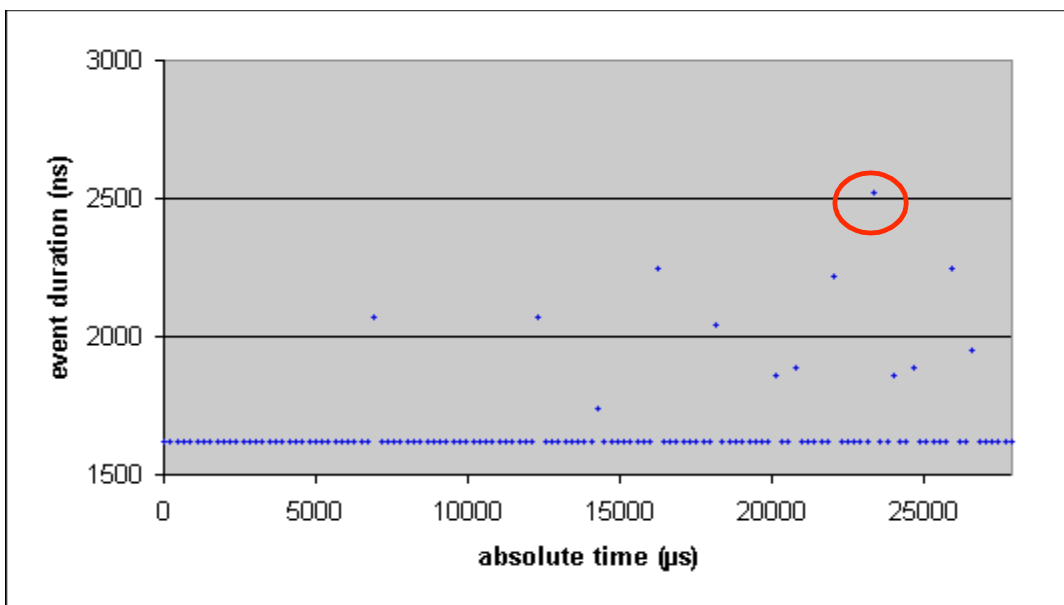


Figure 4.2-7 : Interrupt latency for the HIGH priority ISR

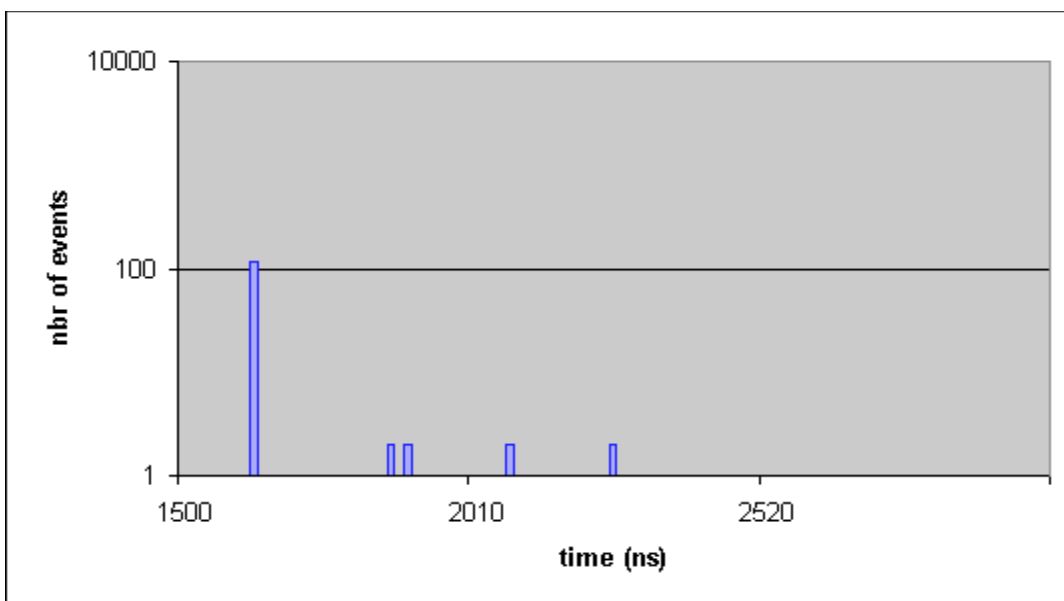


Figure 4.2-8: Interrupt latency for the HIGH priority ISR – Frequency Distribution

4.2.3.2 Interrupt latency – ISR_LO

- Number of threads: 2
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	4.0	4.9	4.1

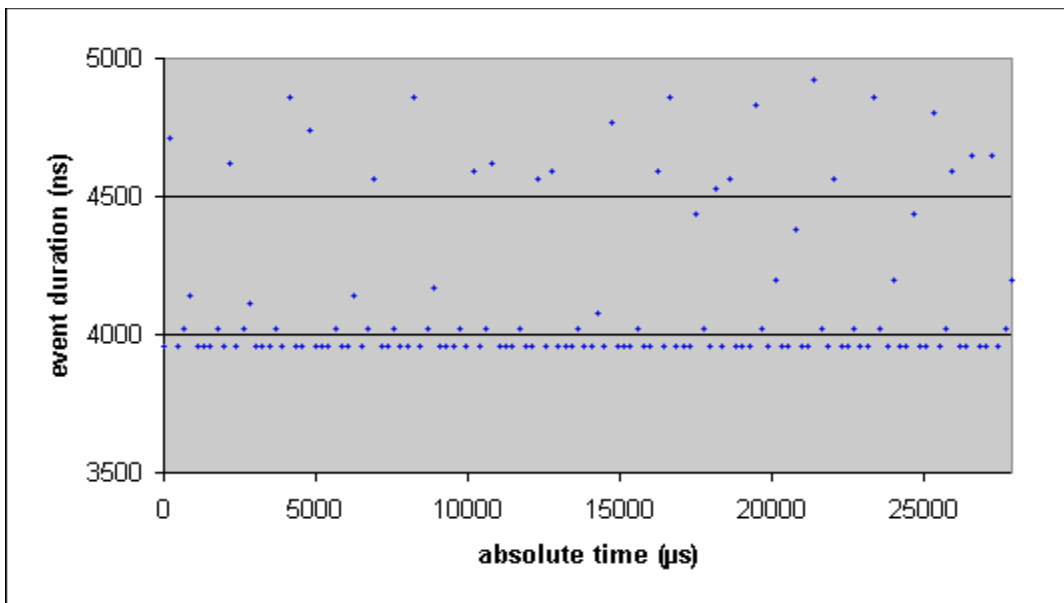


Figure 4.2-9: Interrupt latency for the LOW priority ISR

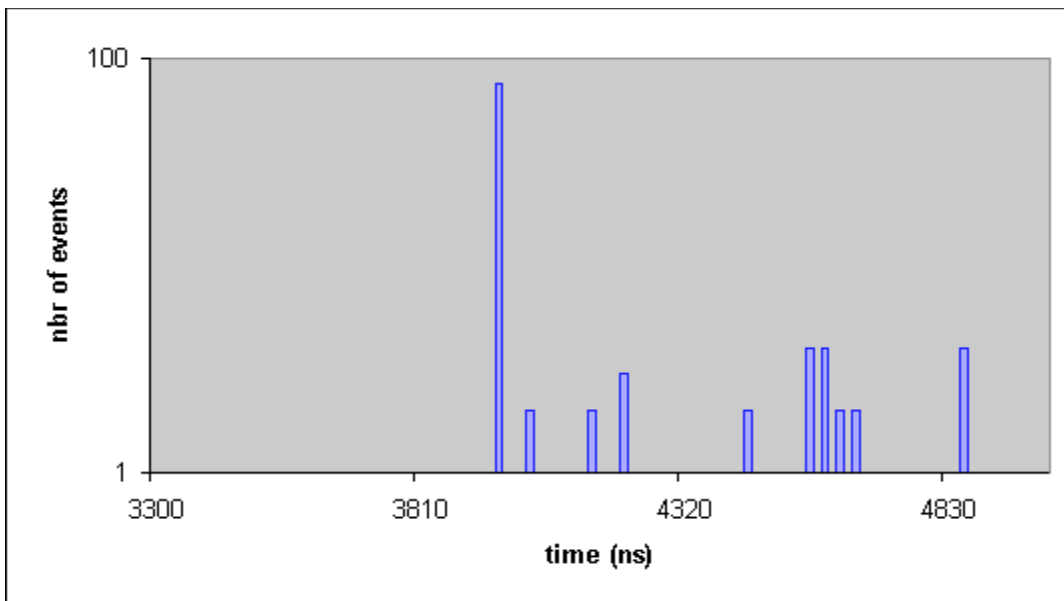


Figure 4.2-10: Interrupt latency for the LOW priority ISR – Frequency Distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.4 Nested interrupt handling

In section 4.2.3, two (nearly) simultaneous interrupts were generated. Both interrupts were generated in a narrow time window, before the system got a chance to act on the first one.

In this section, the same test will be repeated but with a small delay between both interrupts. The purpose is to generate the high priority interrupt while the low priority interrupt is already being handled by its ISR. A good RTOS should be able to handle nested interrupts without any problems.

The following tests will vary the delay between the low and high interrupt from 1.5 μ s to 3 μ s.

4.2.4.1 Nested Interrupt handling – 1.5 μs interval

First PDrive_LO generates the low priority interrupt (INTB#, IRQ 10), followed 1.5μs later by the higher priority interrupt generated by PDrive_HI (INTC#, IRQ 9).

The results in sections 4.2.4.1.1 and 4.2.4.1.2 show that all the high-priority interrupts were serviced before their low-priority counterparts. The bus analyzer snapshot in Figure 4.2-11 demonstrates how IRQ_LO occurs before ISR_HI (signal INTB# drops to active-low before INTC#), while IRQ_HI ends before IRQ_LO (signal INTC# goes back to inactive before INTB#).

☺ This is already a good indication that interrupts can be nested, but is not yet watertight proof. After all, it is possible that IRQ_HI occurred before IRQ_LO was being serviced by ISR_LO. We did however verify by inspecting the analyzer's traces that ISRs can indeed be nested.

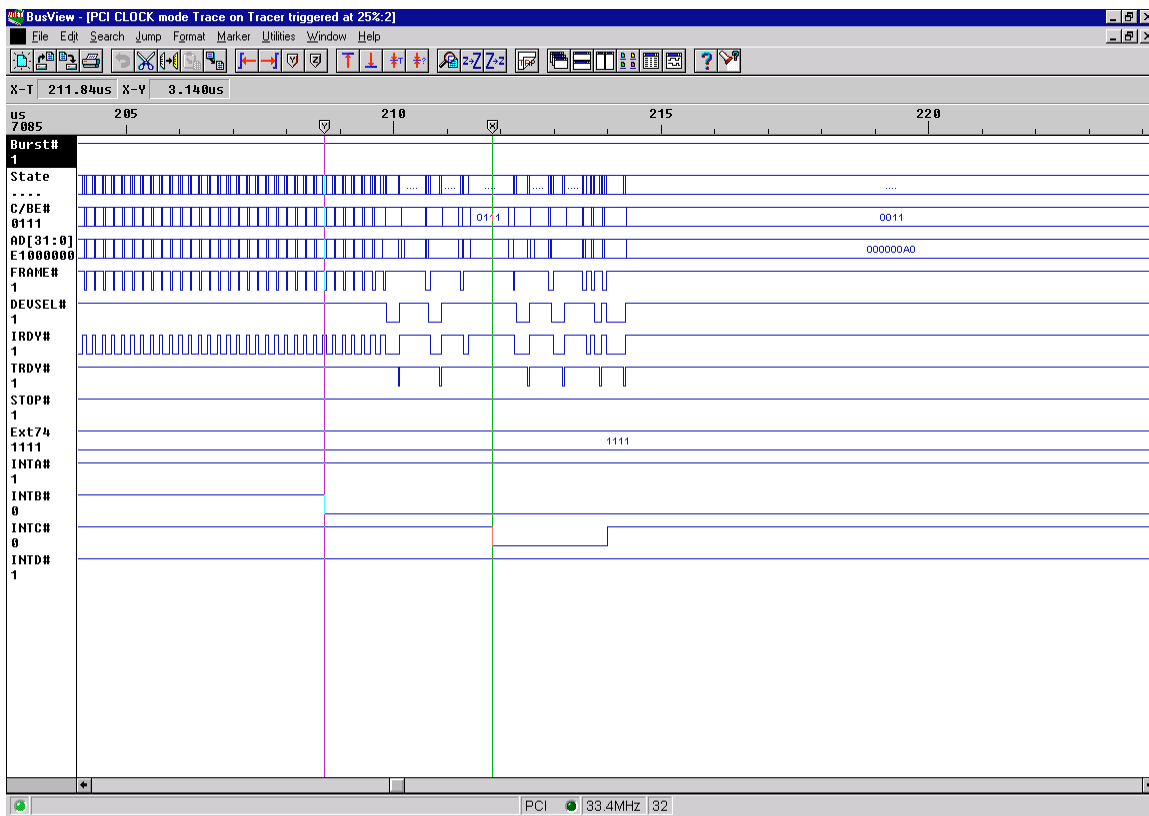


Figure 4.2-11: PCI bus analyzer screenshot

4.2.4.1.1 Interrupt Latency – ISR_HI

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	1.6	3.8	2.7

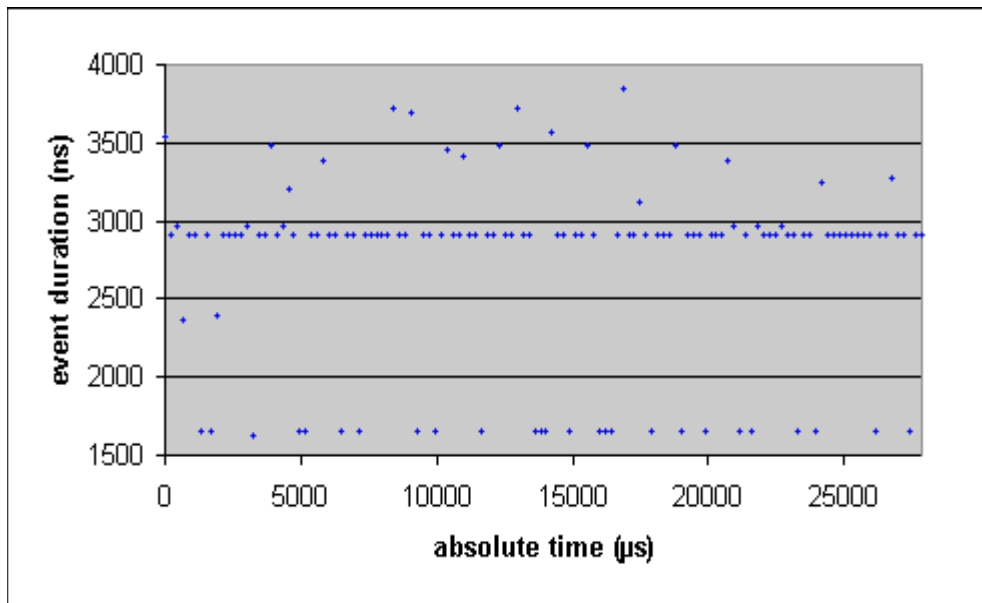


Figure 4.2-12: Interrupt Latency for HIGH priority ISR

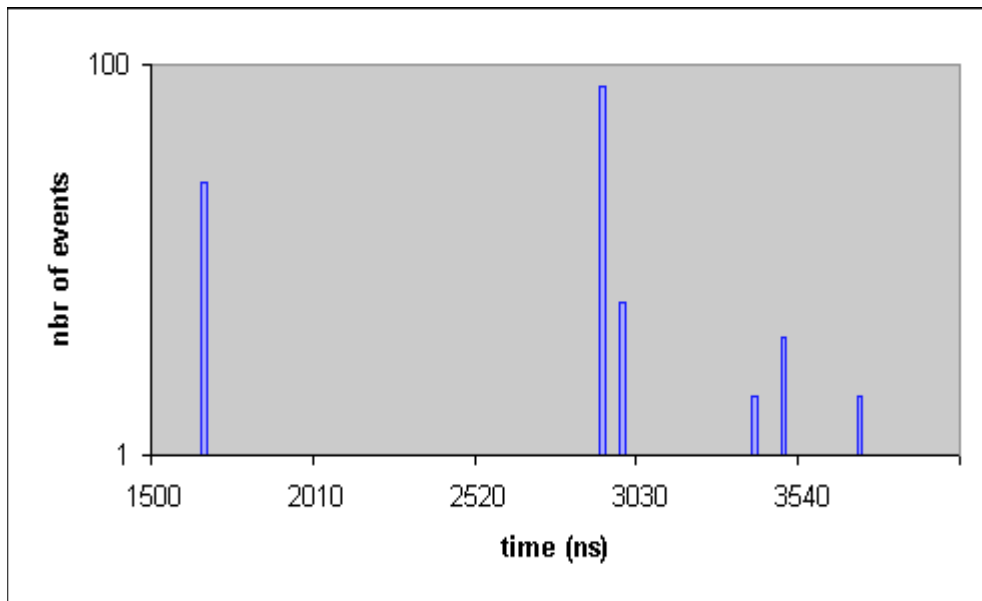


Figure 4.2-13: Interrupt Latency for HIGH priority ISR – Frequency Distribution

4.2.4.1.2 Interrupt Latency – ISR_LO

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	4.0	5.1	4.2

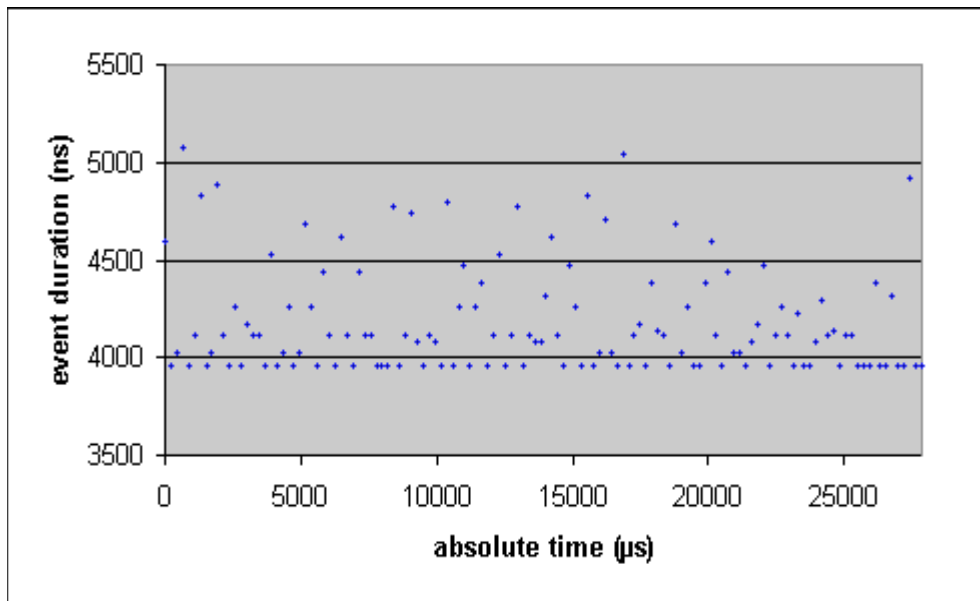


Figure 4.2-14: Interrupt Latency for LOW priority ISR

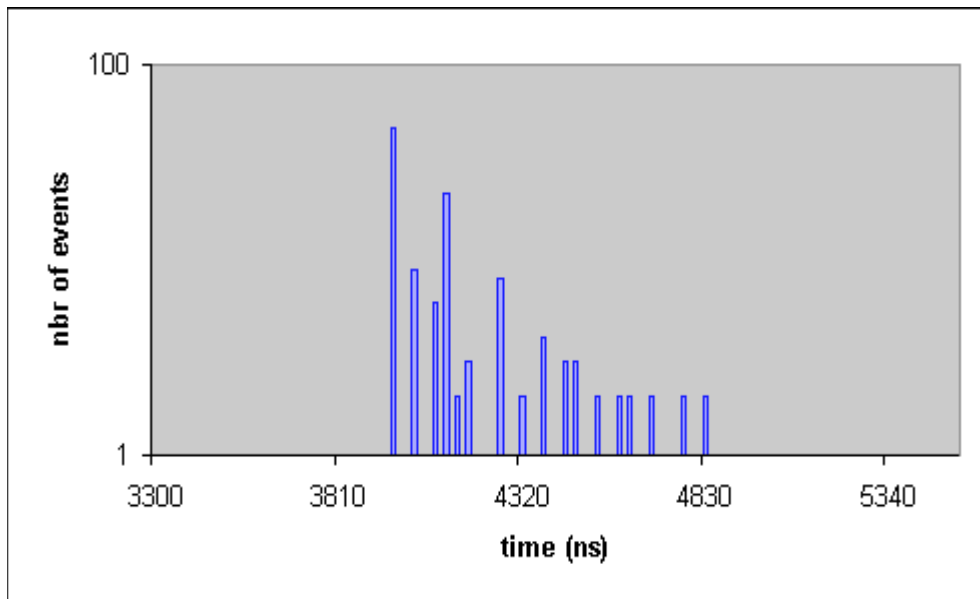


Figure 4.2-15: Interrupt Latency for LOW priority ISR – Frequency Distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.4.2 Nested Interrupt handling – 2.25 μ s interval

This is the same test as in the previous section, with the time interval increased to 2.25 μ s. The results show that in the majority of the case ISR_LO executes before ISR_HI.

4.2.4.2.1 Interrupt Latency – ISR_HI

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	1.6	5.0	3.8

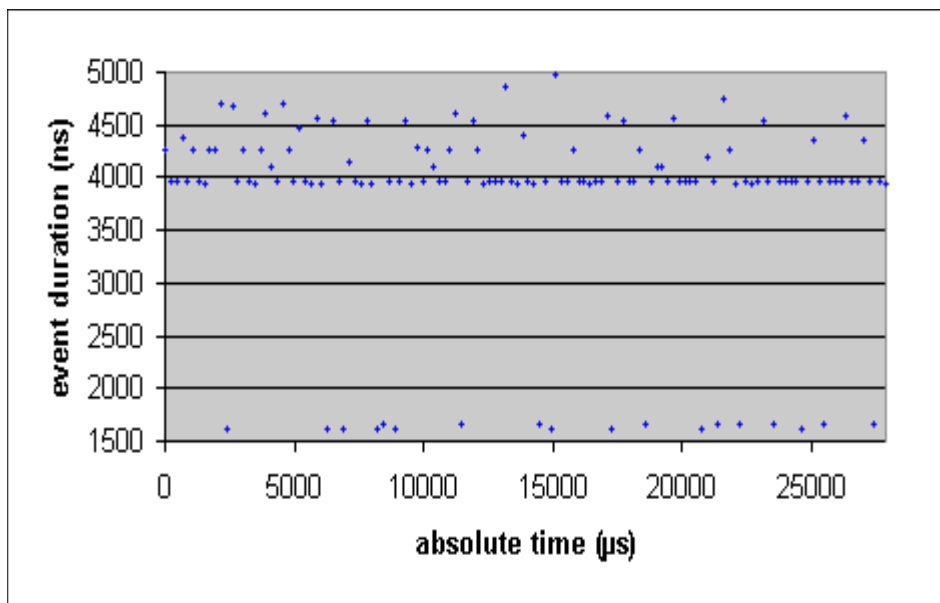


Figure 4.2-16: Interrupt Latency for HIGH priority ISR

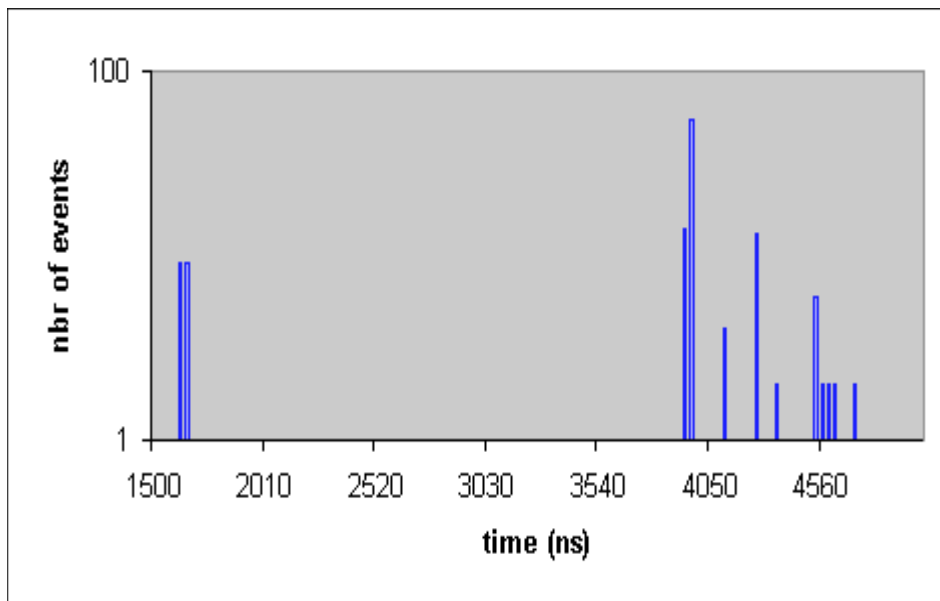


Figure 4.2-17: Interrupt Latency for HIGH priority ISR – Frequency Distribution

4.2.4.2.2 Interrupt Latency – ISR_LO

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	2.9	3.9	3.0

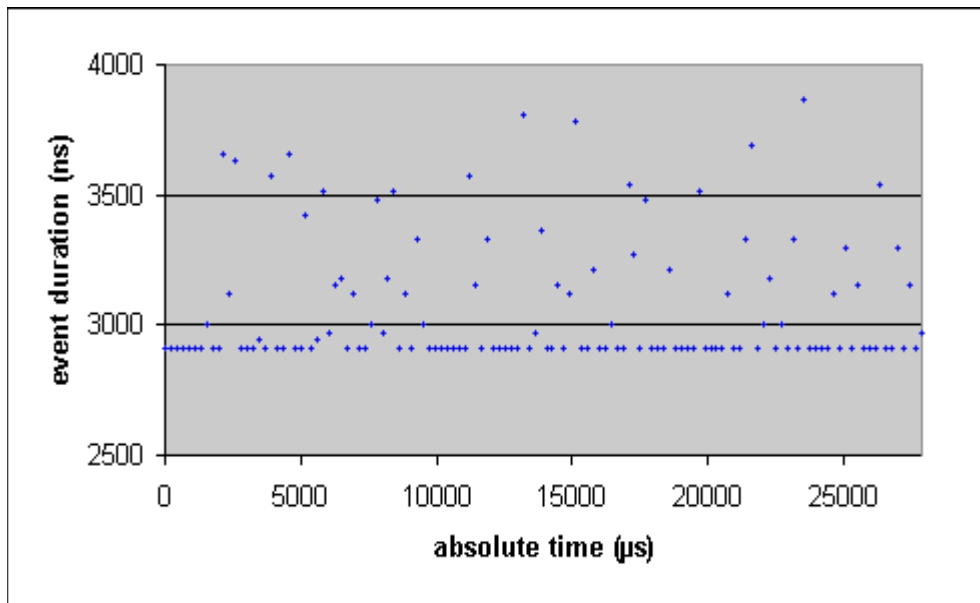


Figure 4.2-18: Interrupt Latency for LOW priority ISR

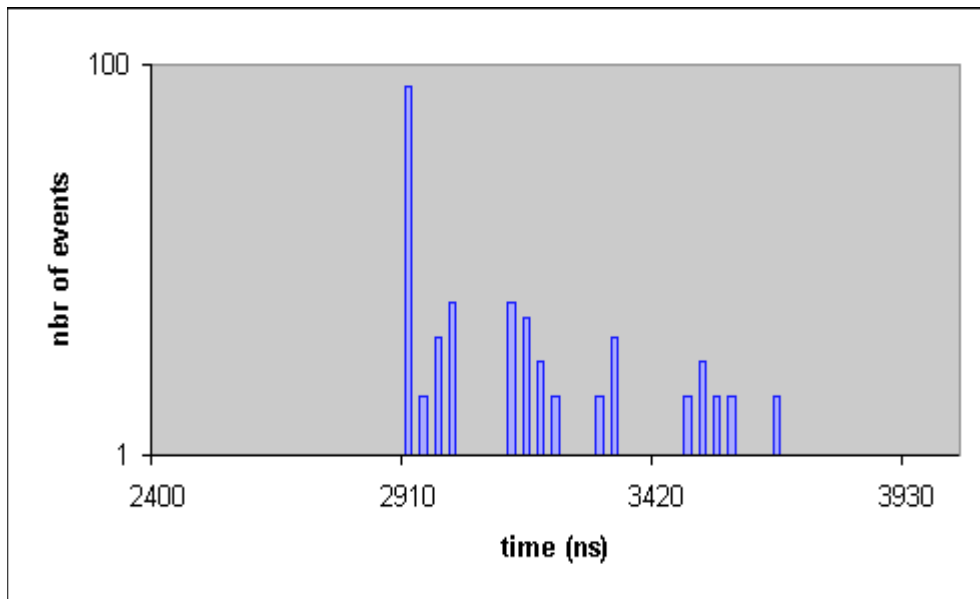


Figure 4.2-19: Interrupt Latency for LOW priority ISR – Frequency Distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.4.3 Nested Interrupt handling – 3 μ s interval

When the interval is increased to 3 μ s, ISR_LO always starts before ISR_HI, because the maximum latency of ISR_LO is smaller than the minimum latency of ISR_HI.

4.2.4.3.1 Interrupt Latency – ISR_HI

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	3.2	7.3	3.8

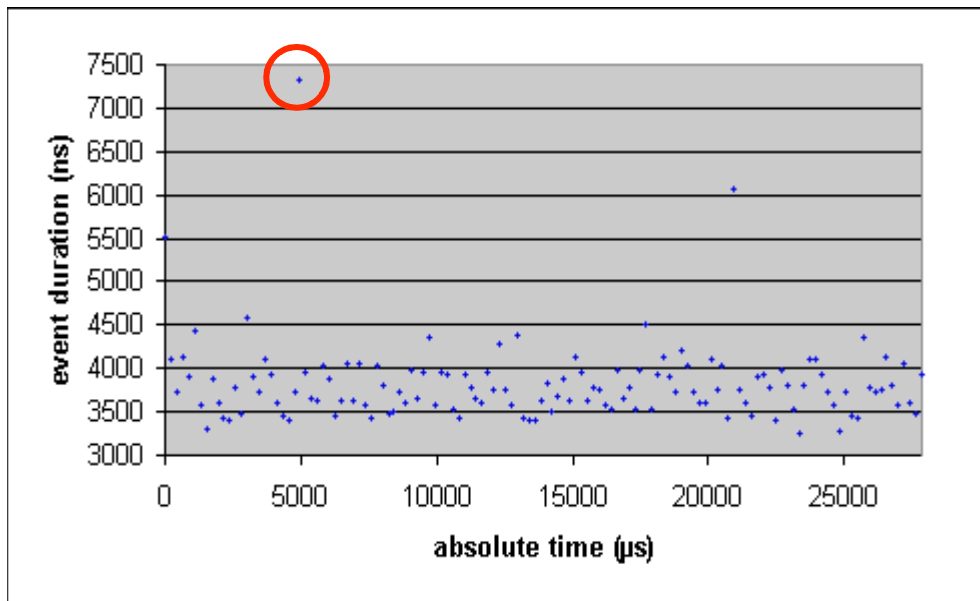


Figure 4.2-20: Interrupt Latency for HIGH priority ISR

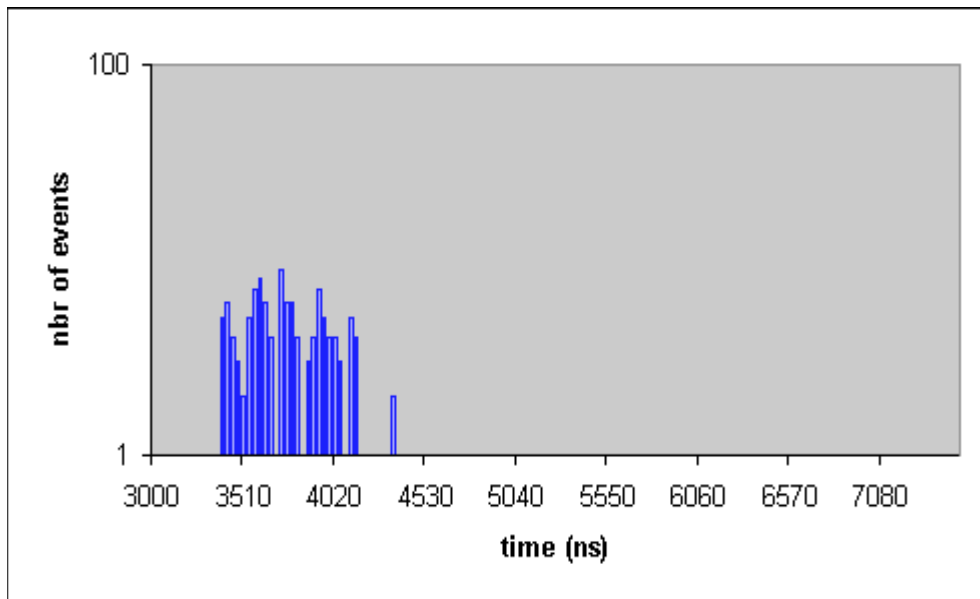


Figure 4.2-21: Interrupt Latency for HIGH priority ISR – Frequency Distribution

4.2.4.3.2 Interrupt Latency – ISR_LO

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
130	1.6	6.8	1.9

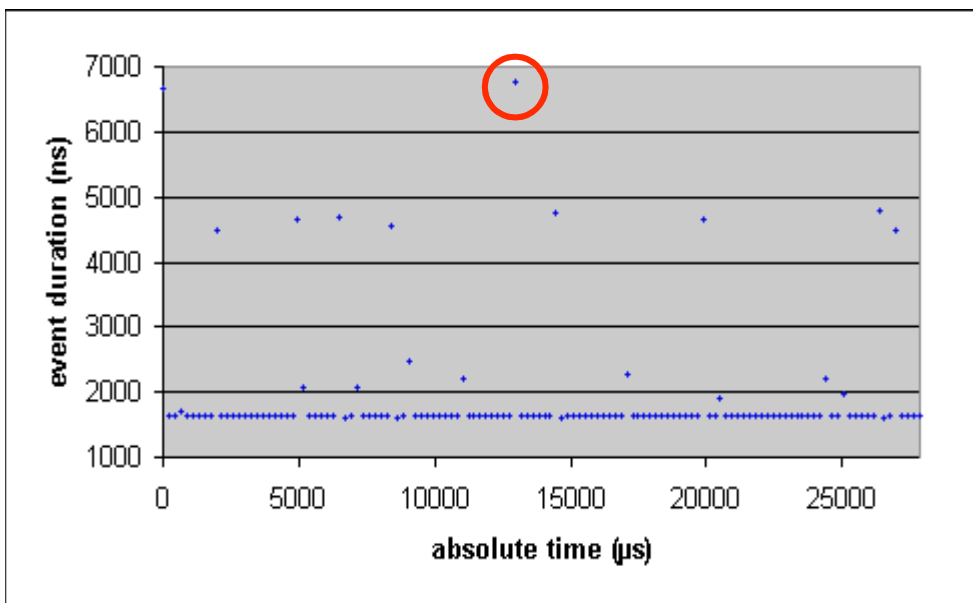


Figure 4.2-22: Interrupt Latency for LOW priority ISR

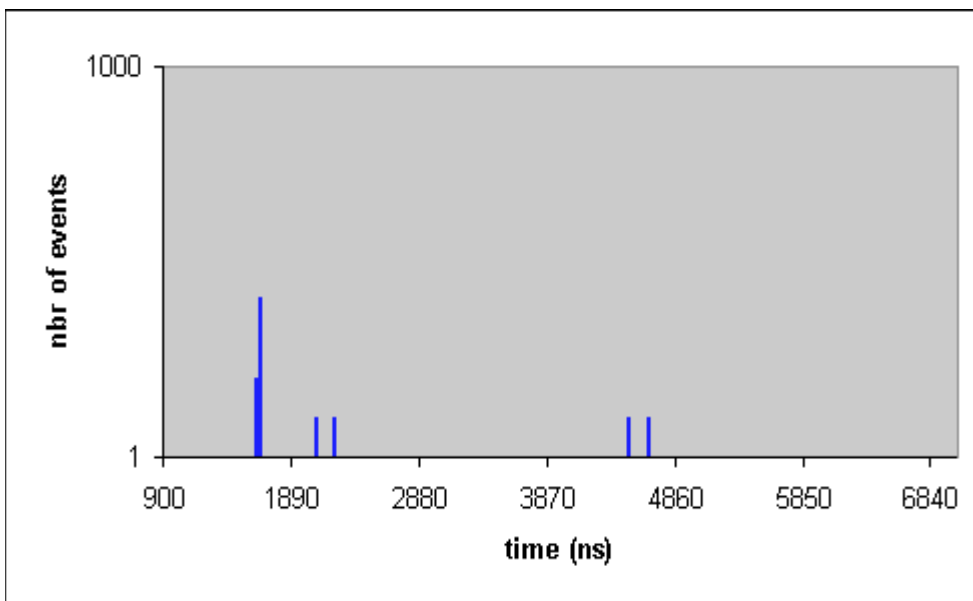


Figure 4.2-23: Interrupt Latency for LOW priority ISR – Frequency Distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.2.5 Maximum sustainable interrupt frequency – ISR level

The purpose of the tests in this section is to determine the maximum sustainable (periodic) interrupt frequency without losing any interrupts.

In section 4.2.5.1, the minimum sustainable time interval between interrupts is determined by executing short tests that generate 1000 periodic interrupts. The test will be repeated with increasing frequencies until one or more interrupts are lost. This way the maximum sustainable interrupt frequency can be accurately determined.

To verify whether or not the results found in section 4.2.5.1 can be sustained over longer periods of time, an endurance test will be performed in section 4.2.5.2, which exposes the system to a BILLION (10^9) periodic interrupts. The test is successful only if it finishes without having lost any interrupts.

All interrupts are serviced on ISR level. The ISR does not perform any useful work, it simply writes a trace to the bus analyzer and returns.

4.2.5.1 Short tests

Table 4.2-1 shows that when interrupts are generated with a period of less than 5.5µs, the system inevitably starts “losing” interrupts.

The maximum interrupt frequency isn’t changed from the previous version of QNX (6.1).

Period (µs)	#interrupts serviced	#interrupts lost
15	1000	0
12	1000	0
9	1000	0
7	1000	0
5.5	1000	0
5	998	2
4	996	4
3	685	315

Table 4.2-1: 1000 periodic interrupts generated at various frequencies

4.2.5.2 Endurance tests

Table 4.2-2 shows the results for the test where the system is exposed to one billion (10^9) periodic interrupts at a particular frequency. Due to the nature of the test set-up, it was not possible to determine the exact amount of interrupts that were lost if that amount exceeded 32768.

The maximum interrupt frequency isn't changed from the previous version of QNX (6.1).

Period (μ s)	#interrupts serviced	#interrupts lost
10	1,000,000,000	0
9	1,000,000,000	0
8	999,999,988	12
7	999,999,799	201

Table 4.2-2: One BILLION interrupts generated at various frequencies

☺ The test succeeded when the interrupt period was set at 9μ s i.e., every single interrupt was serviced. When the interrupt period was brought down to 8μ s, 12 interrupts did not get serviced.

The reader should bear in mind that this test does not only stress test the RTOS, but also the PC hardware. So no hard conclusions can be drawn as to which part of the system is to blame for "losing" interrupts. However, as this test is done in all our tests with the same hardware, results between the different RTOS evaluations may be compared.

4.3 Threads

The test in this section handles thread creation and deletion. The test continuously creates and deletes a thread. The threads are created in a joined state and the creating thread joins the thread on deletion.

In QNX the thread deletion is handled by the thread being deleted itself, and thus at the priority level of the thread being deleted. In most RTOSes it is the thread deleting the thread that will free the resources. In fact the QNX way of doing this is more predictable, as a high priority thread deleting a low priority thread does not have to do all the handling (which is in fact a kind of priority inversion!).

Therefore we had to adapt our test a bit to get comparable results: the deleting thread waits on the deleted thread (using a join), to be sure that all resources are freed. Without such a wait, the system uses all it's memory: indeed the deleted threads never become active, so the resources are not freed. It's notable that in this case the thread creation call returns an error, but the system itself stays alive.

4.3.1 Creation (TF-a-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
6553	172	2880	175

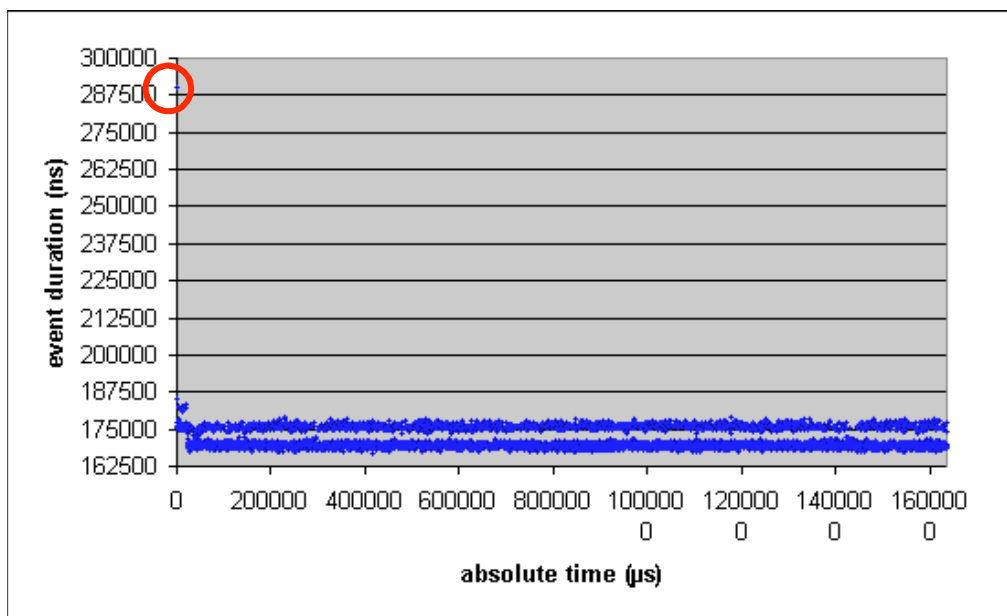


Figure 4.3-1 TF-a-1.a

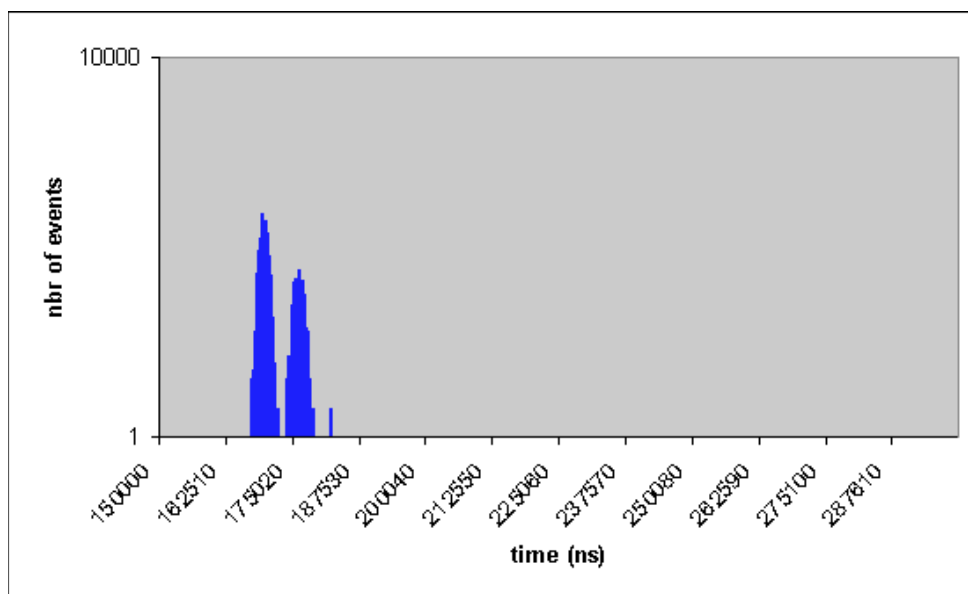


Figure 4.3-2 TF-a-1.a - frequency distribution

4.3.2 Deletion (TF-b-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
6553	76.5	102	78

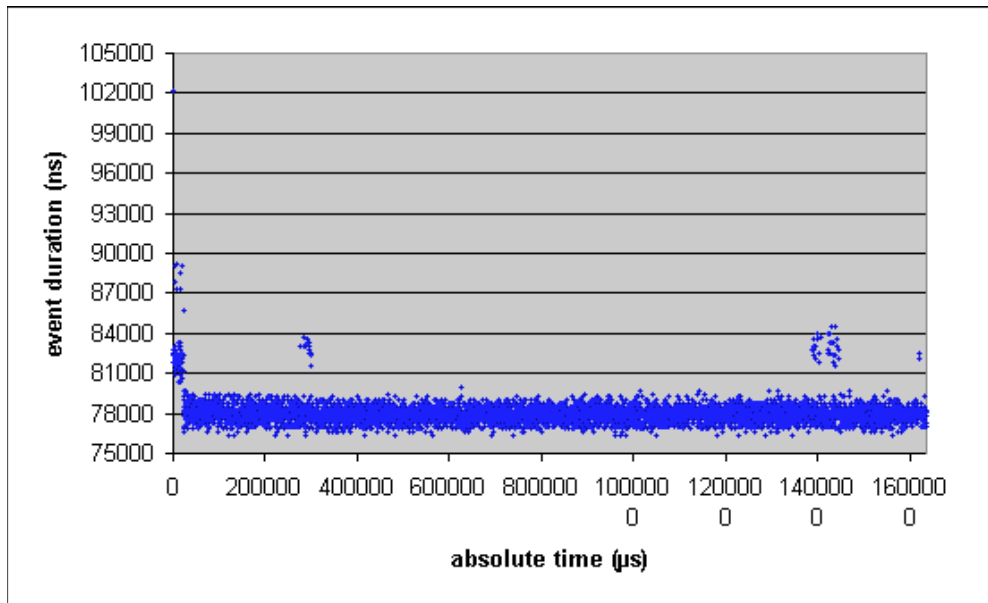


Figure 4.3-3 TF-b-1.a

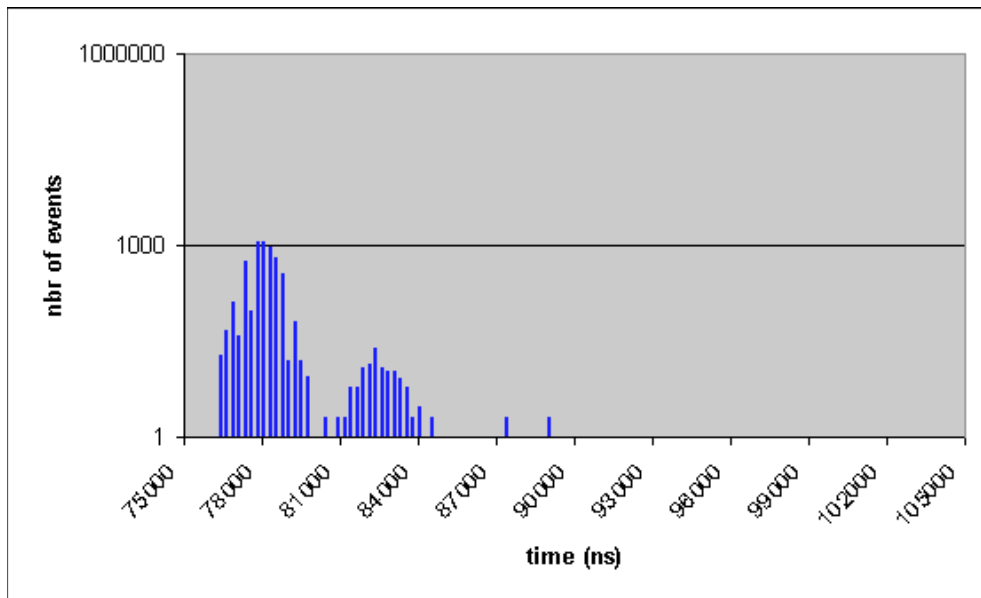


Figure 4.3-4 TF-b-1.a - frequency distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.4 Thread switch latency – same process

For this test, a number of threads of equal priority are created. As soon as one thread becomes active, it automatically yields the processor to let another thread run. The test is repeated with 2, 10 and 128 threads. The thread switch latency is the time the system needs to switch from one thread to another. All the threads run in the same process.

4.4.1 Thread switch latency (TSL-a-2.d)

- Number of threads: 2
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	2.5	8.3	2.6

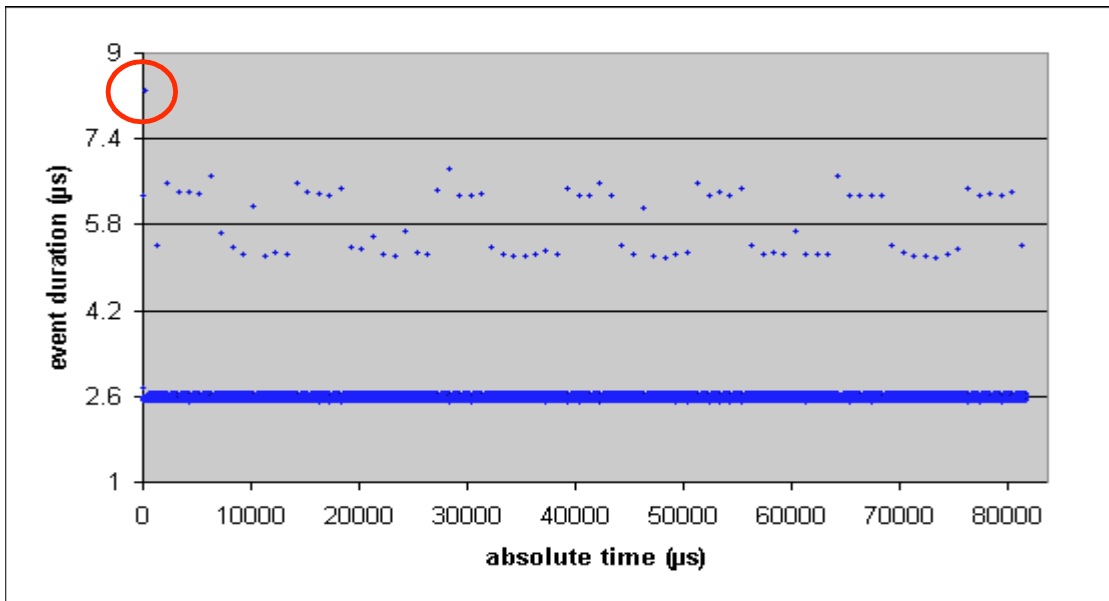


Figure 4.4-1 TSL-a-2.d

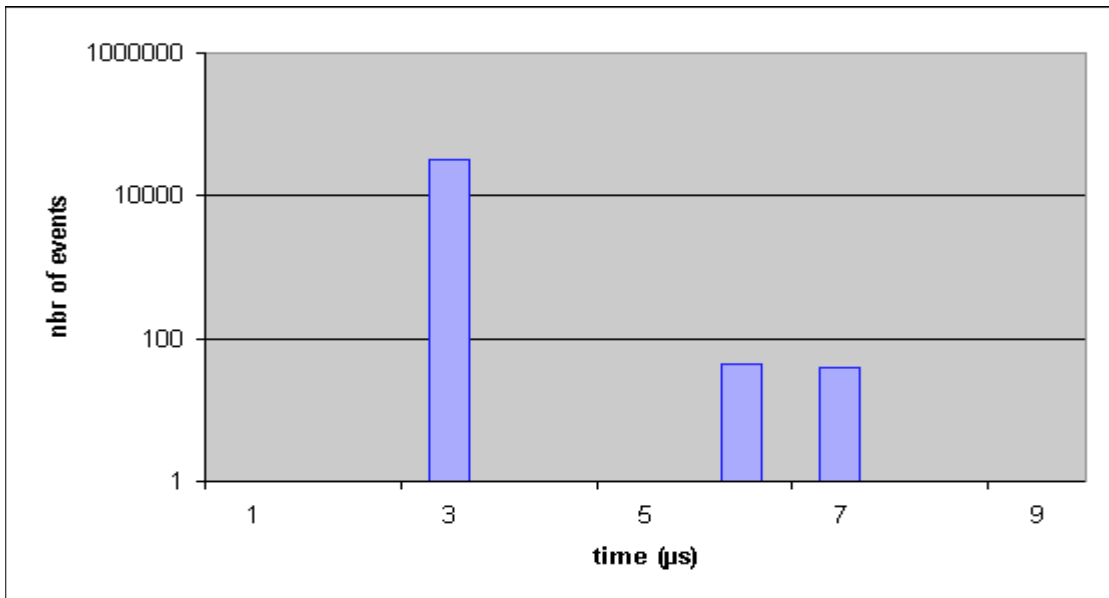


Figure 4.4-2 TSL-a-2.d - frequency distribution

4.4.2 Thread switch latency (TSL-a-10.d)

- Number of threads: 10
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	2.8	7.9	2.8

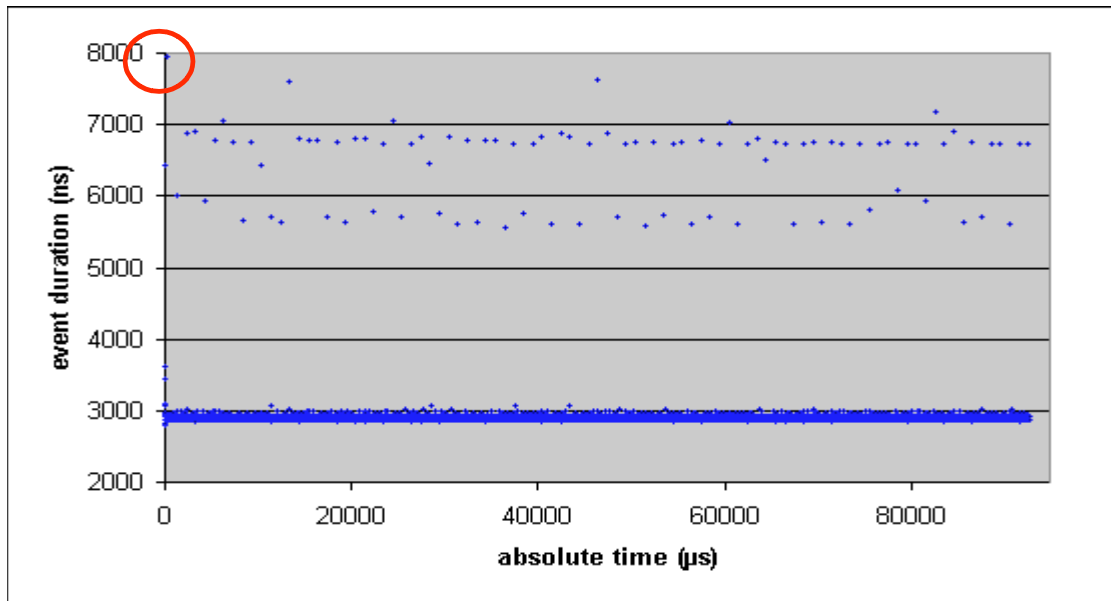


Figure 4.4-3 TSL-a-10.d

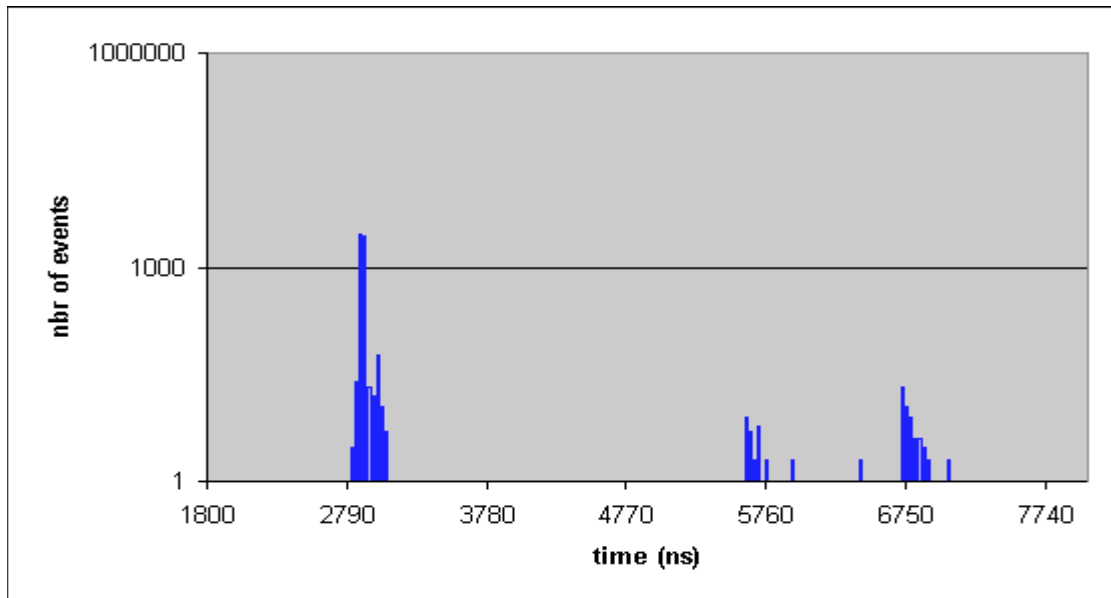


Figure 4.4-4 TSL-a-10.d - frequency distribution

4.4.3 Thread switch latency (TSL-a-128.d)

- Number of threads: 128
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	2.9	9.5	3.6

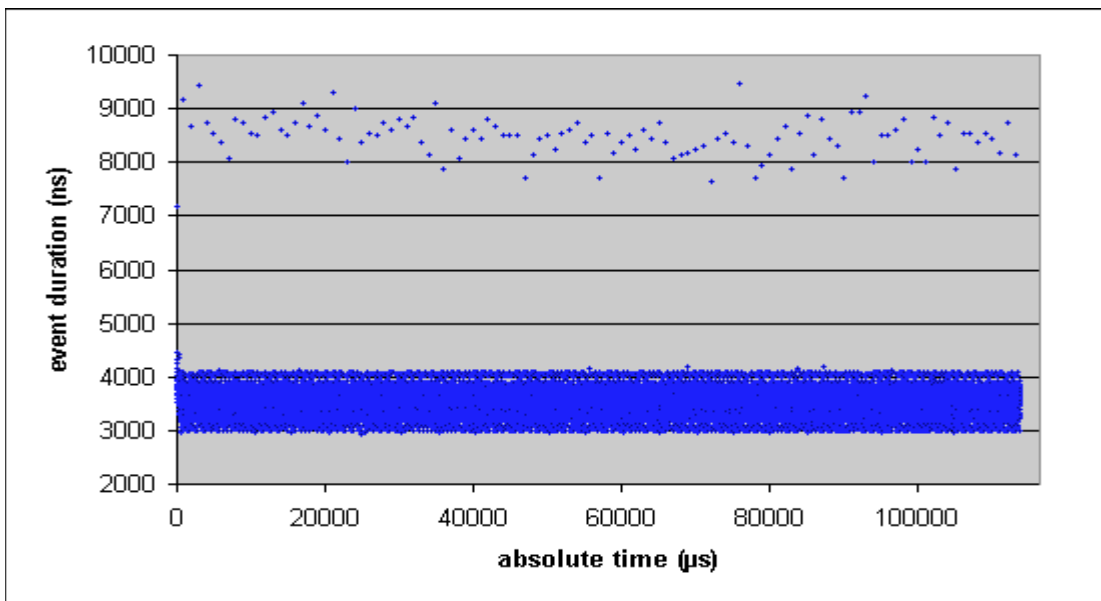


Figure 4.4-5 TSL-a-128.d

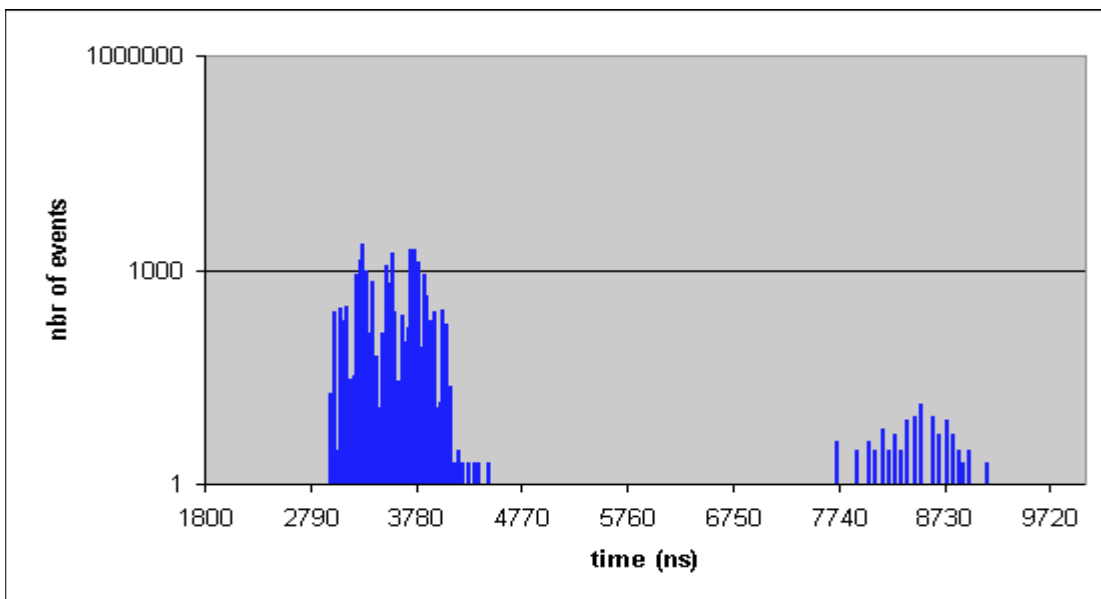


Figure 4.4-6 TSL-a-128.d - frequency distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.5 Thread switch latency – different process

This test is basically the same as the thread switch latency test in section 4.4, except that every thread in this system belongs to a different process. So this test measures the time it takes to switch from one process to another.

The test was executed with 2, 10 and 128 processes. Switching between processes is slower because every process has its own virtual memory space. However, the worst-case process switch latency we encountered was still less than 22µs.

The worst case scenario happens when a process is first activated; this can easily be seen on the tests with 128 processes.

4.5.1 Thread switch latency (TSL-b-2.d)

- Number of threads: 2
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	5.0	11.2	5.2

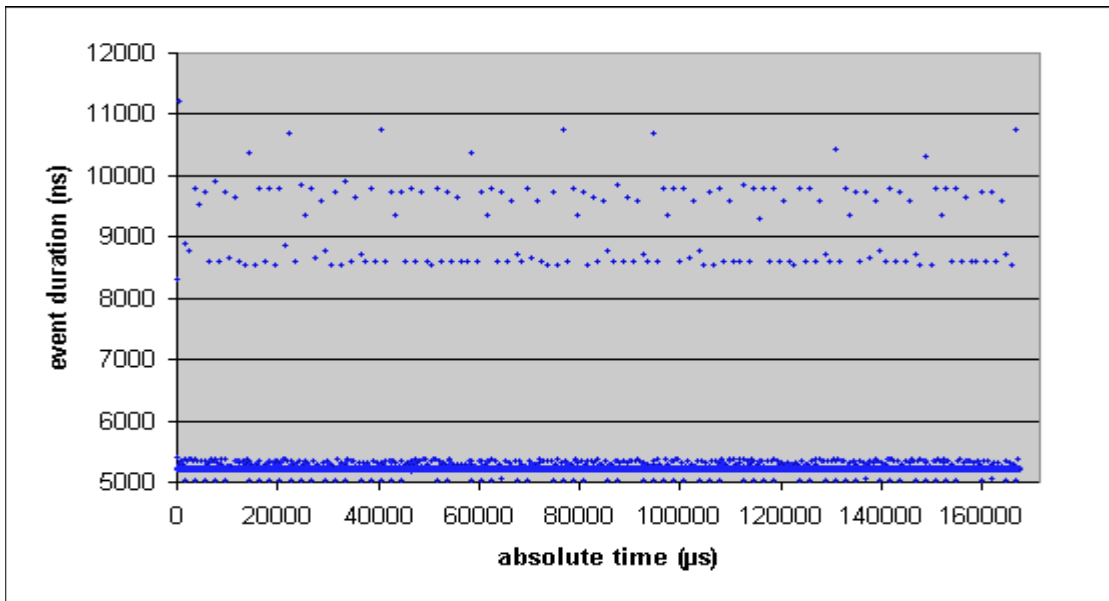


Figure 4.5-1 TSL-b-2.d

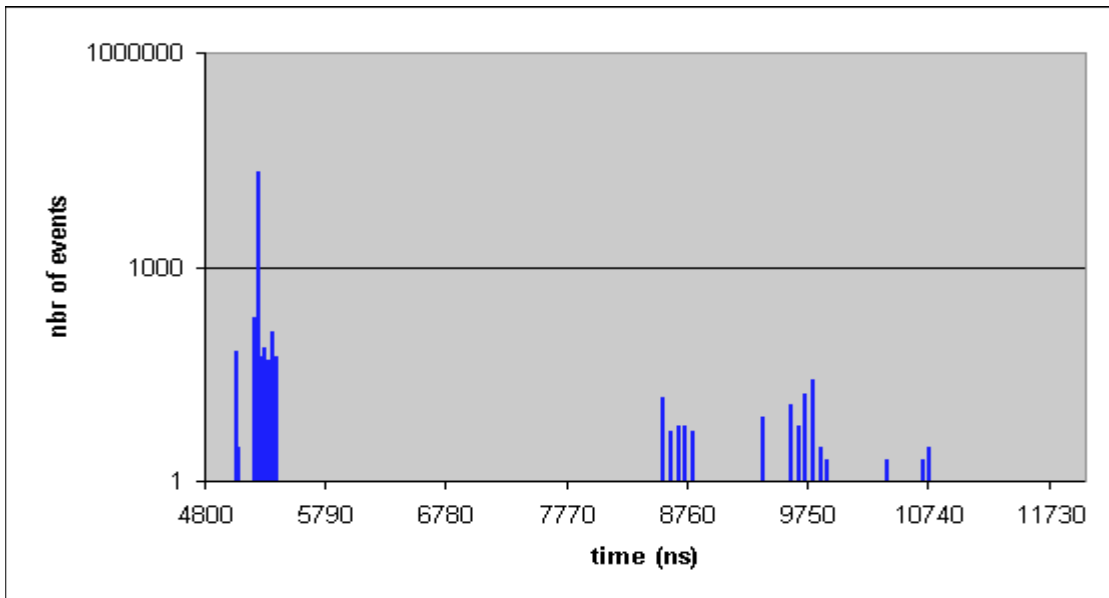


Figure 4.5-2 TSL-b-2.d - frequency distribution

4.5.2 Thread switch latency (TSL-b-10.d)

- Number of threads: 10
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	5.5	12.0	5.9

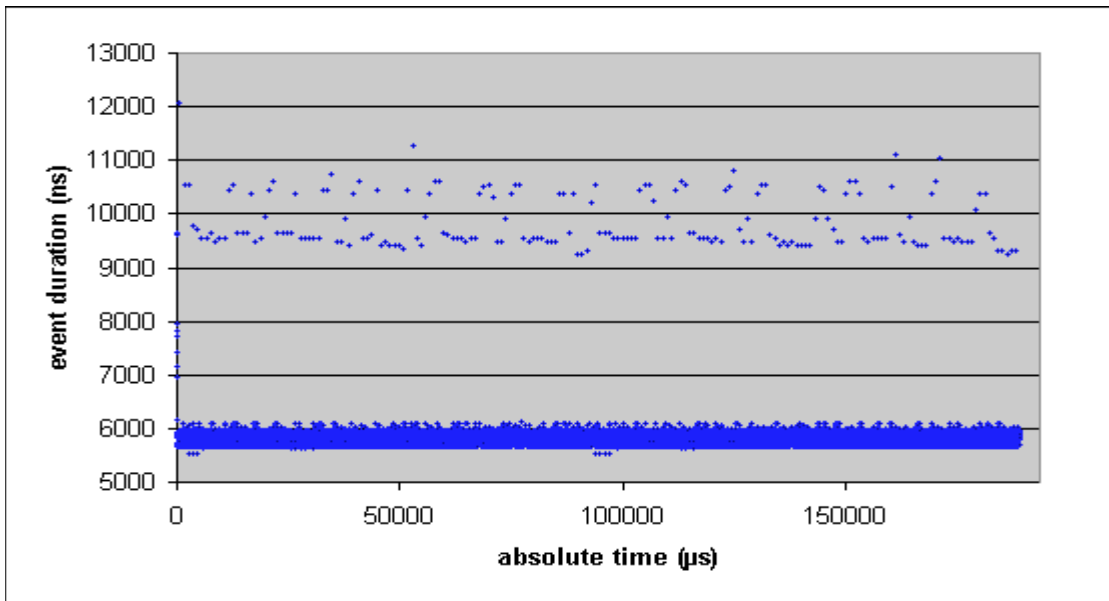


Figure 4.5-3 TSL-b-10.d

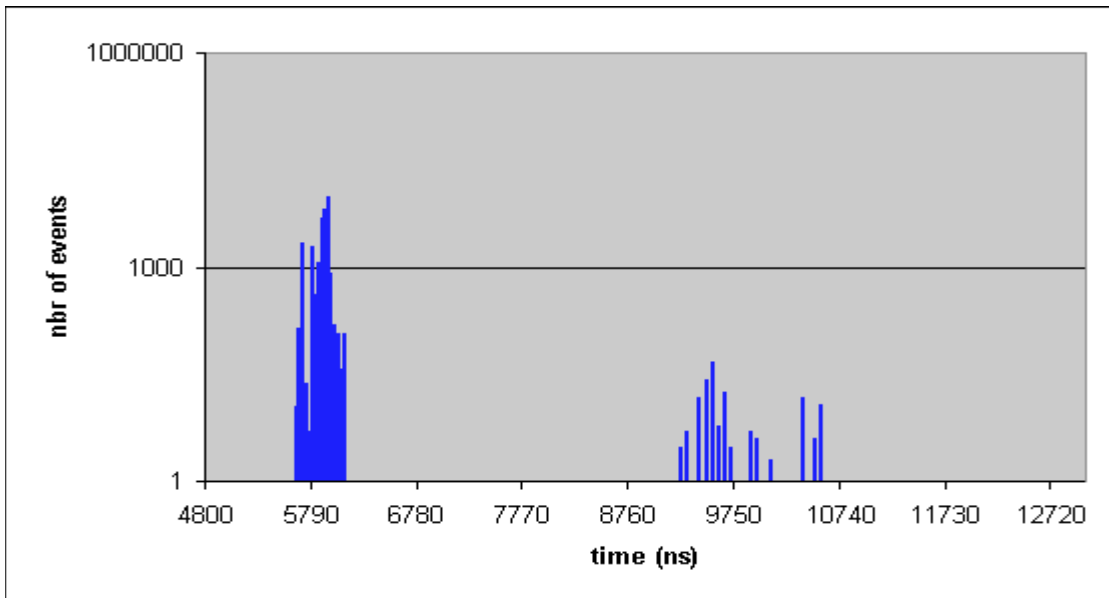


Figure 4.5-4 TSL-b-10.d - frequency distribution

4.5.3 Thread switch latency (TSL-b-128.d)

- Number of threads: 128
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
32766	6.7	21.8	8.8

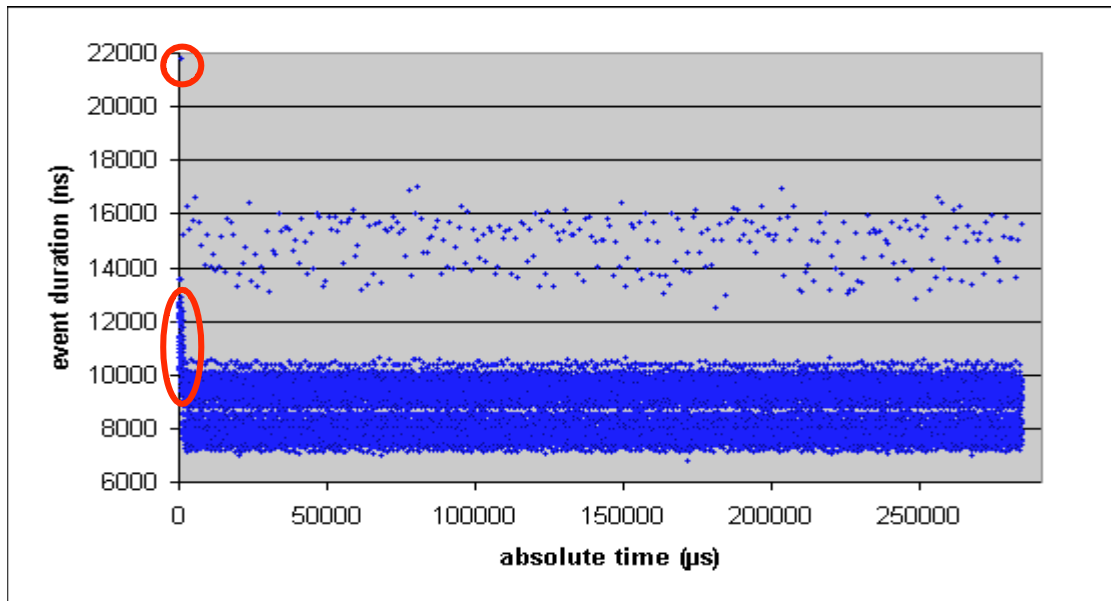


Figure 4.5-5 TSL-b-128.d

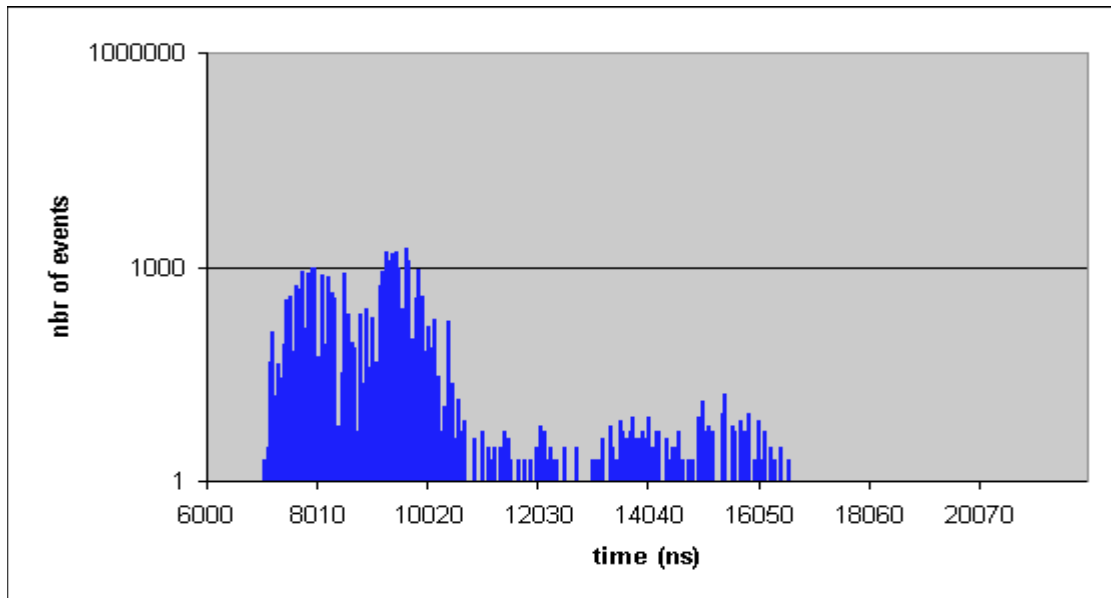


Figure 4.5-6 TSL-b-128.d - frequency distribution

4.6 Counting semaphores

The tests in sections 4.6.1, 4.6.2 and 4.6.3 handle the semaphore creation and deletion time. The peaks in the test results are due to the clock interrupt occurring every millisecond during our measurements. It can also be observed that there is no difference between the time needed to delete a semaphore that hasn't been used and a semaphore that has been used. This suggests that all the necessary resources are allocated when the call to create the semaphore is executed.

The synchronization test, for which the results are displayed in section 4.6.6, creates a situation where a number of threads with different priorities are pending on the same semaphore. The QNX NEUTRINO RTOS v6.2 has 64 different priority levels. Since lowest priority level is reserved for the idle task and cannot be used by applications, the test was executed with 63 threads.

☺ The idea of this test is to check if the release + rescheduling time is independent of the number of threads pending on the semaphore. This is the case in the QNX NEUTRINO RTOS v6.2, as can be seen from Figure 4.6-12. The measured metric remains constant during one test cycle as the number of threads waiting on the semaphore ramps up from 1 to 63.

The release + rescheduling time is independent of the number of threads pending on the semaphore in the QNX NEUTRINO RTOS v6.2. This probably means that the system sorts the queue of threads pending on the semaphore whenever a new thread is added to it. The results for the synchronization test were reprocessed to extract the time it takes for the system to acquire a semaphore that is non-signaled and reschedule to let another thread run (see section 4.6.7) . Whenever a thread tries to acquire a semaphore that is non-signaled, it is added to the semaphore's queue. The purpose of this test is to verify that the time it takes to add a new thread to the semaphore's queue (and sort it!!) is independent of the number of threads in the queue.

☺ The results in Figure 4.6-14 (p.68) show that there is no such dependency between the time it takes to sort the queue and the number of threads in the queue.

4.6.1 Creation (SEO-a-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8192	3.4	8.5	3.4

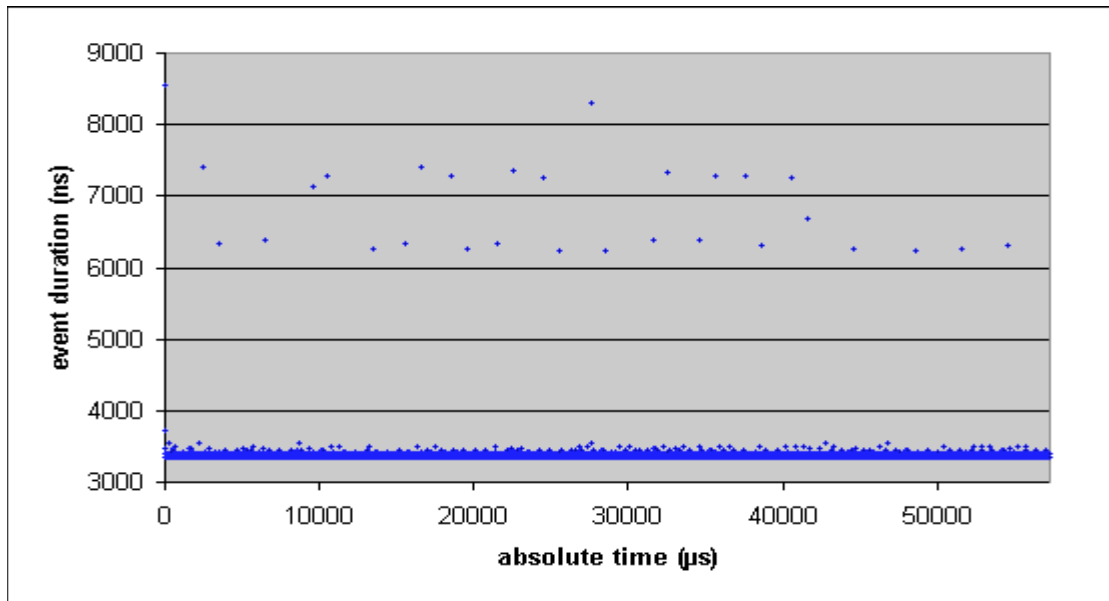


Figure 4.6-1 SEO-a-1.d

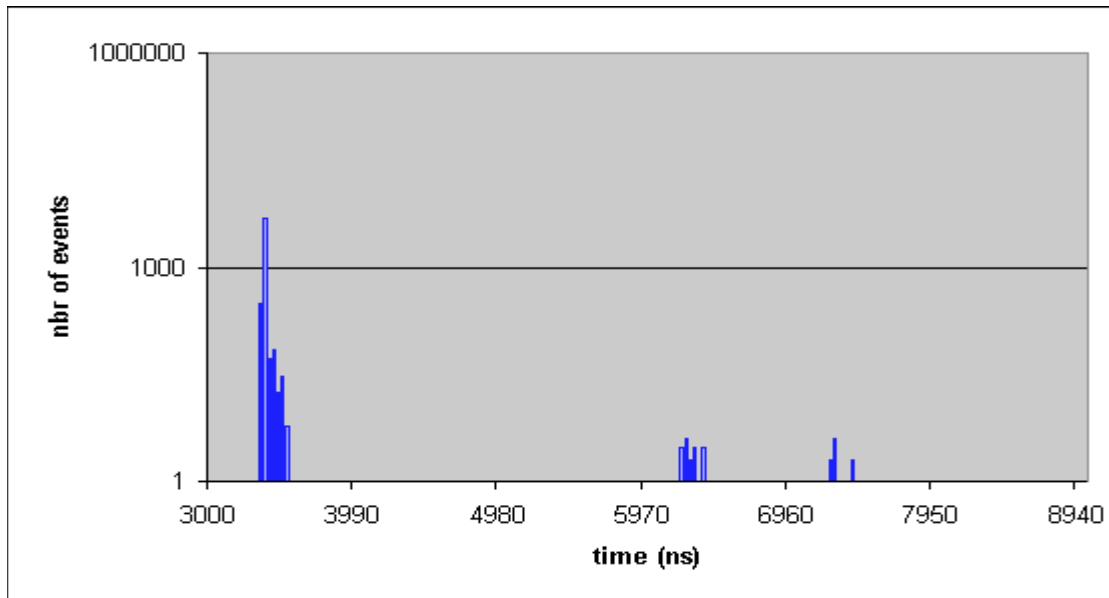


Figure 4.6-2 SEO-a-1.d - frequency distribution

4.6.2 Deletion (SEO-b-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8191	3.2	10.2	3.2

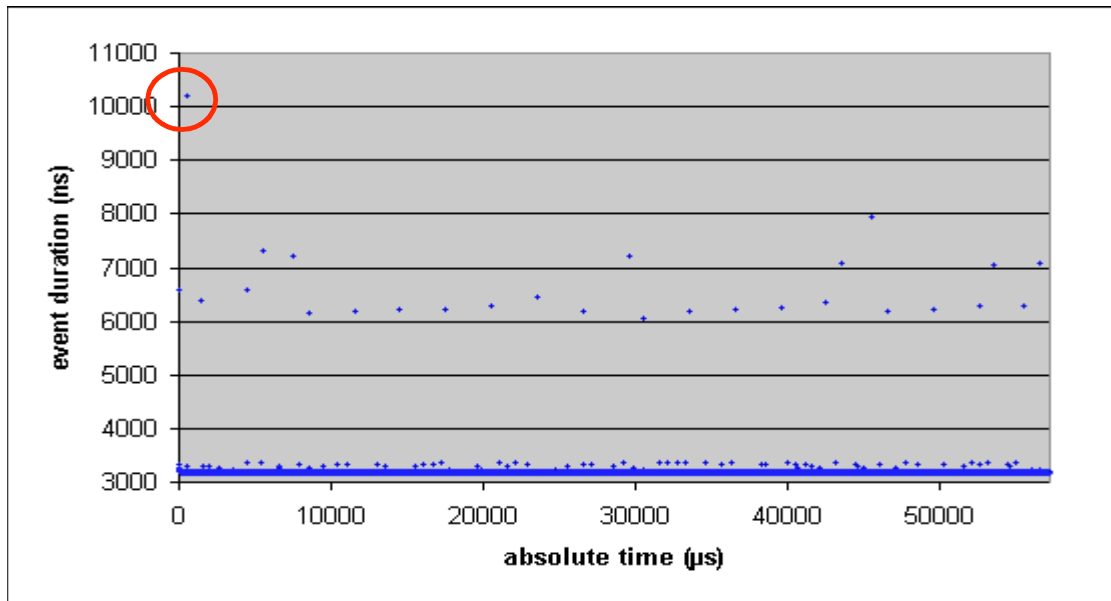


Figure 4.6-3 SEO-b-1.d

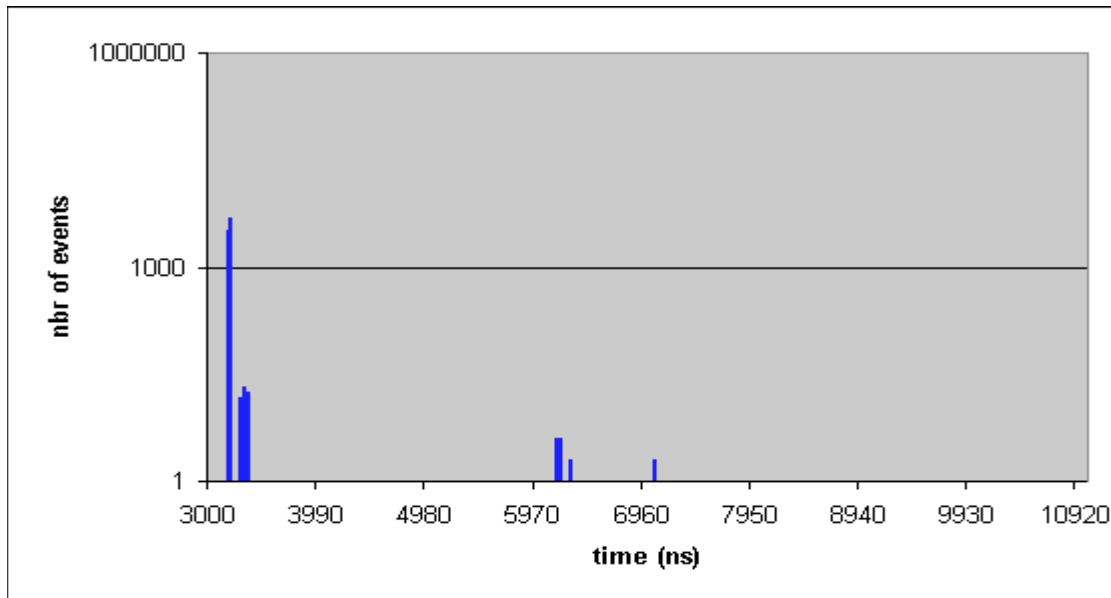


Figure 4.6-4 SEO-b-1.d - frequency distribution

4.6.3 Deletion - after use (SEO-c-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8191	3.3	7.5	3.4

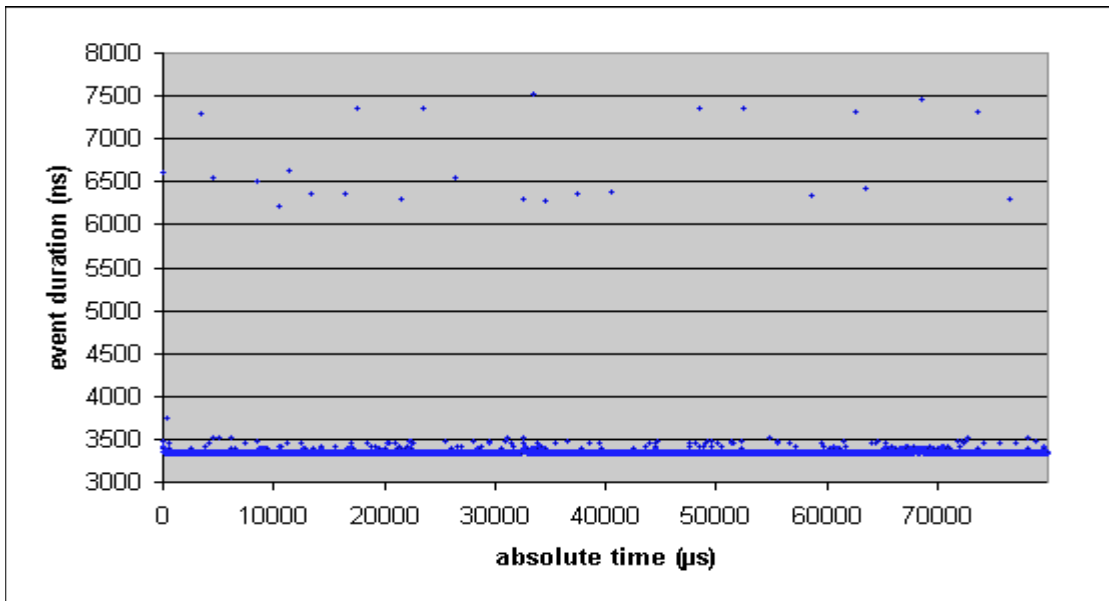


Figure 4.6-5 SEO-c-1.d

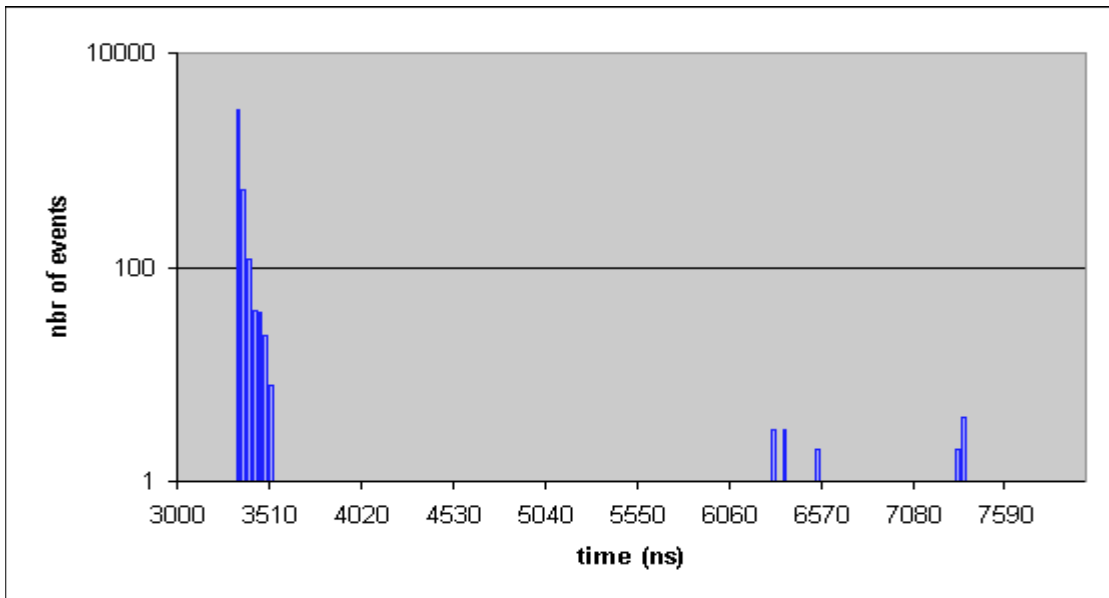


Figure 4.6-6 SEO-c-1.d - frequency distribution

4.6.4 No contention – Acquire (SEO-d-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8192	2.5	9.1	2.5

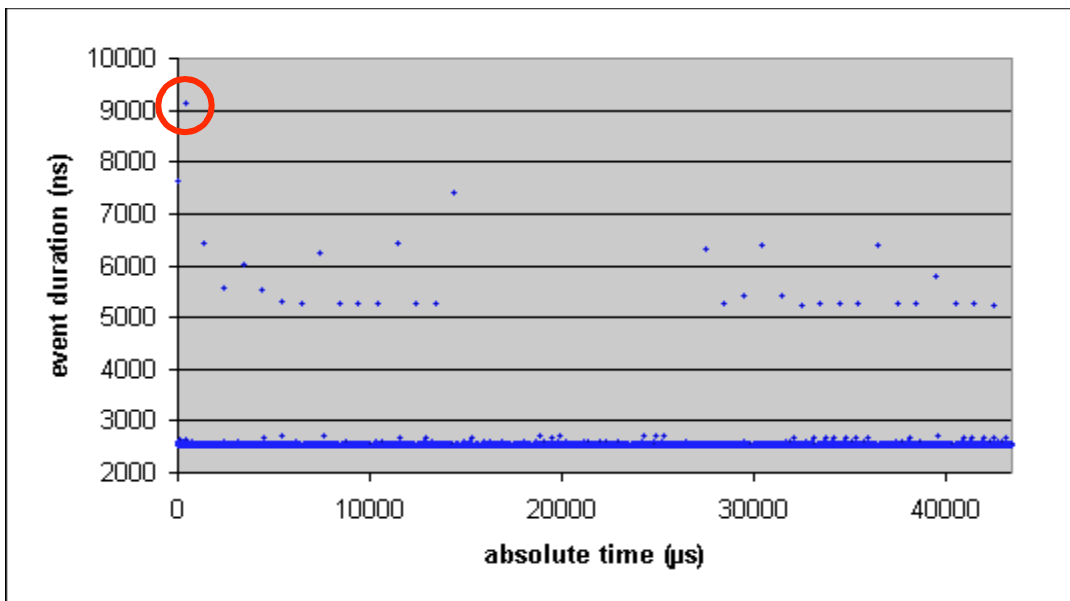


Figure 4.6-7 SEO-d-1.d

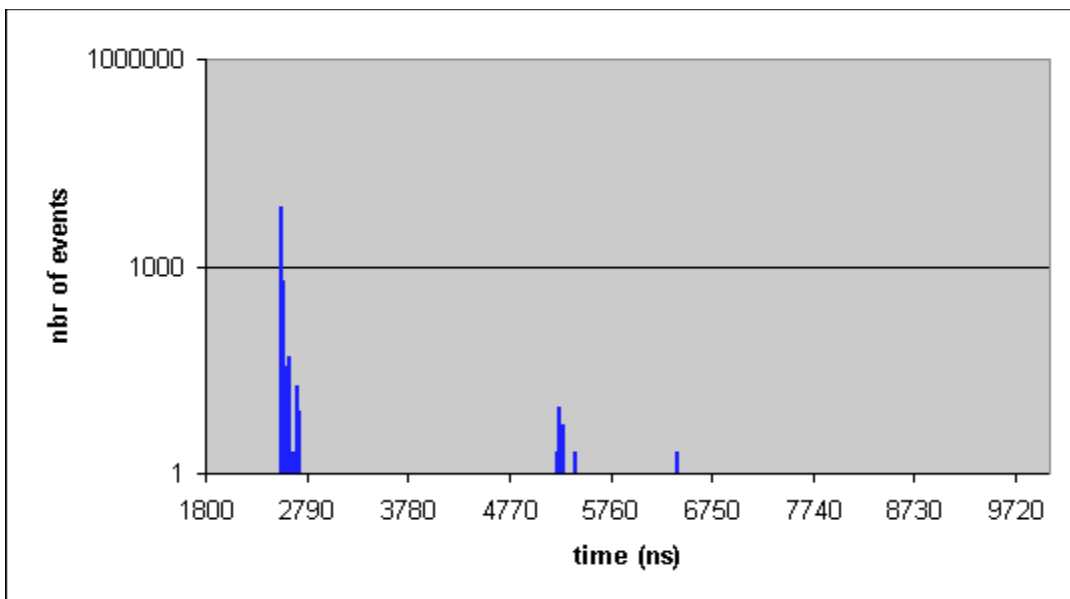


Figure 4.6-8 SEO-d-1.d - frequency distribution

4.6.5 No contention – Release (SEO-e-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8191	2.4	6.4	2.4

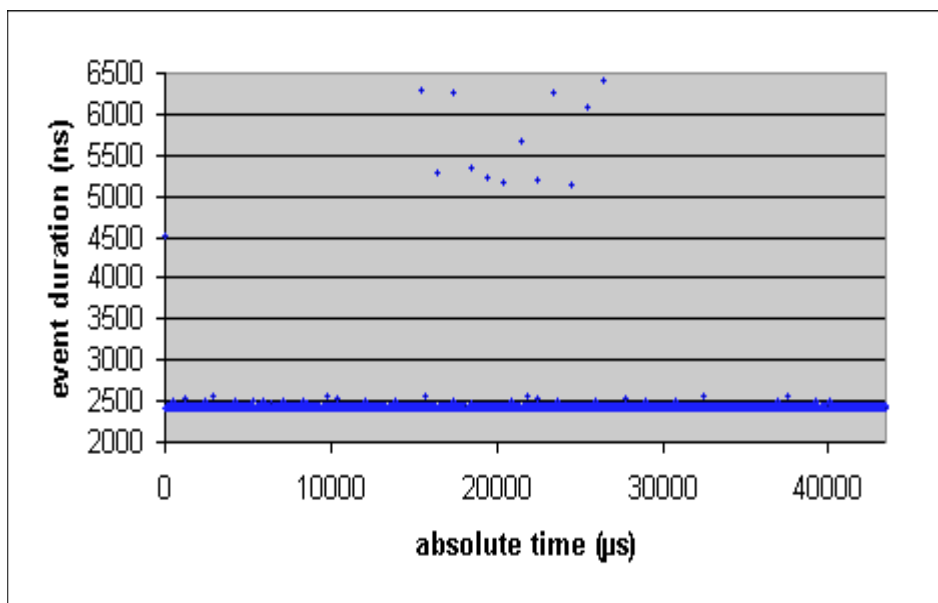


Figure 4.6-9 SEO-e-1.d

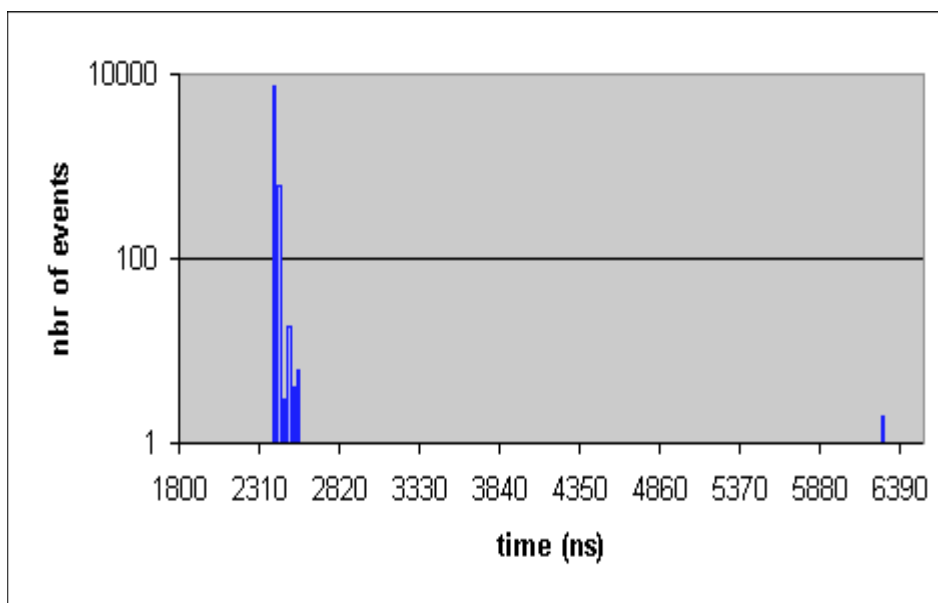


Figure 4.6-10 SEO-e-1.d - frequency distribution

4.6.6 Synchronization (SEO-f-63.d)

- Number of threads: 63
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8273	5.9	12.7	6.6

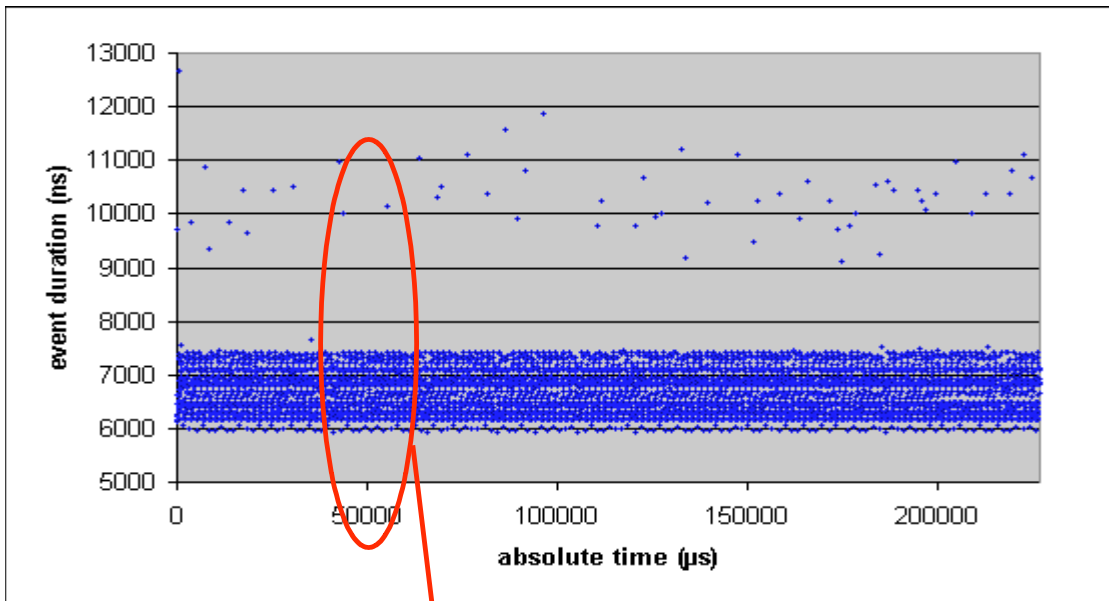


Figure 4.6-11 SEO-f-63.d

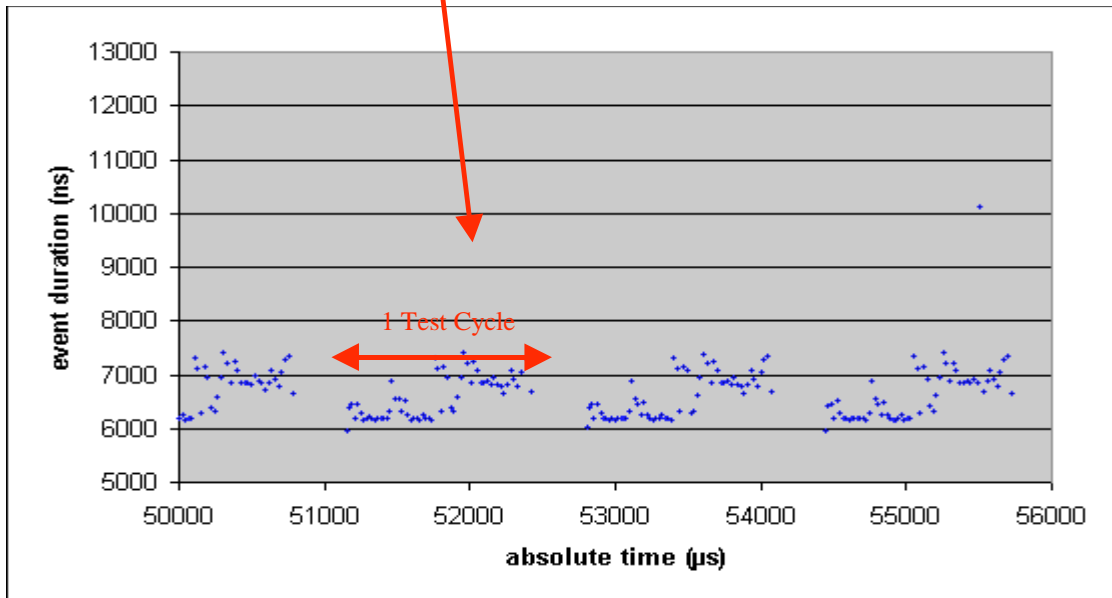


Figure 4.6-12 SEO-f-63.d – Exploded view

4.6.7 Acquiring a semaphore that is non-signaled

- Number of threads: 247
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8273	5.7	11.7	6.7

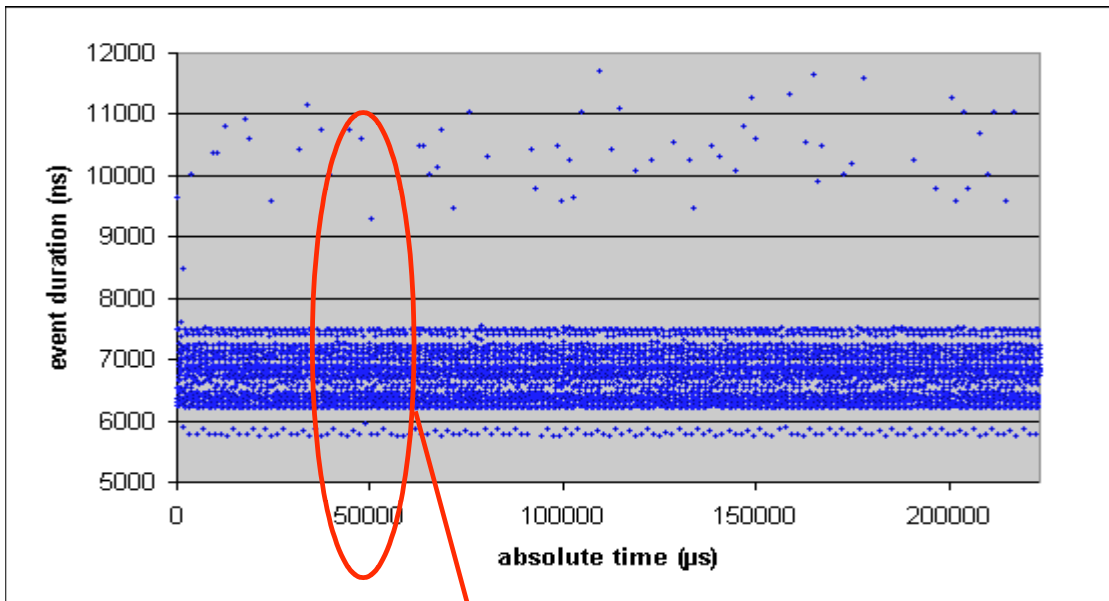


Figure 4.6-13 Acquiring a semaphore that is non-signaled

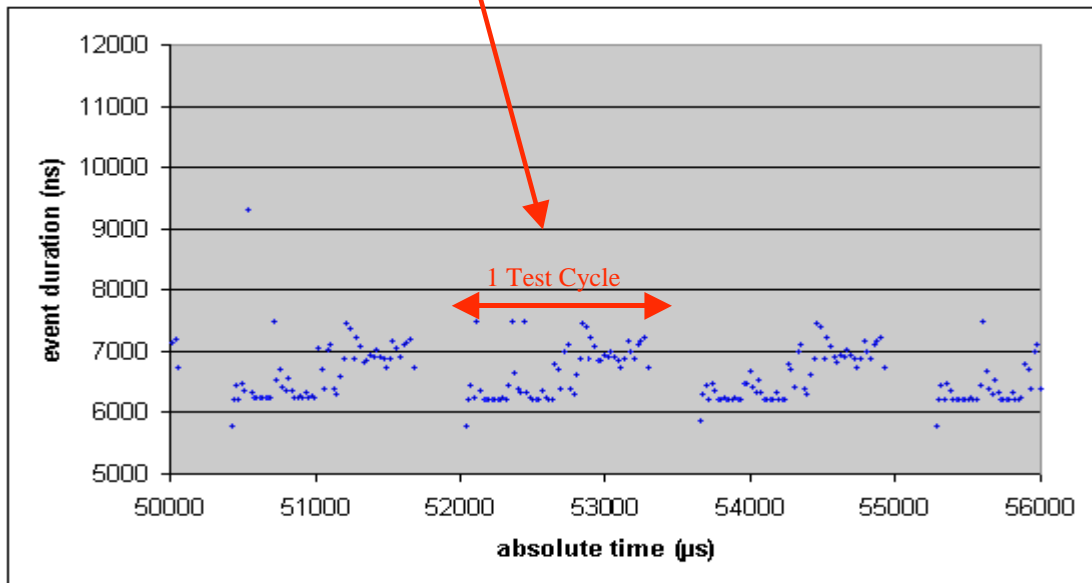


Figure 4.6-14 Acquiring a semaphore that is non-signaled – exploded view

4.7 Mutexes

The test on uncontested acquisition and release of a mutex, for which the results are displayed in sections 4.7.1 and 4.7.2., is implemented as a continuous loop of an acquisition followed immediately by a release of the mutex.

The QNX NEUTRINO RTOS v6.2 has a priority inheritance mechanism, which is 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 the high priority thread can't acquire it. 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 is stable; the maximum time it took to perform this series of actions was less than 14 μ s, as can be seen in Figure 4.7-5 (p. 72). Again, the clock interrupt is the cause for the stray samples.

Compared with the previous version of QNX we tested (6.1), there is a good improvement in this test. Clearly, the priority inversion case is an important worst case scenario that happens sometimes in a real-time system. This important scenario has improved more than 30% compared with the previous release.

Version	Minimum (μ s)	Maximum (μ s)	Average (μ s)
QNX 6.1	11.6	19.0	11.9
QNX 6.2	7.1	13.7	7.4

4.7.1 No contention – Acquire (SEO-d-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8192	0.4	9.6	0.4

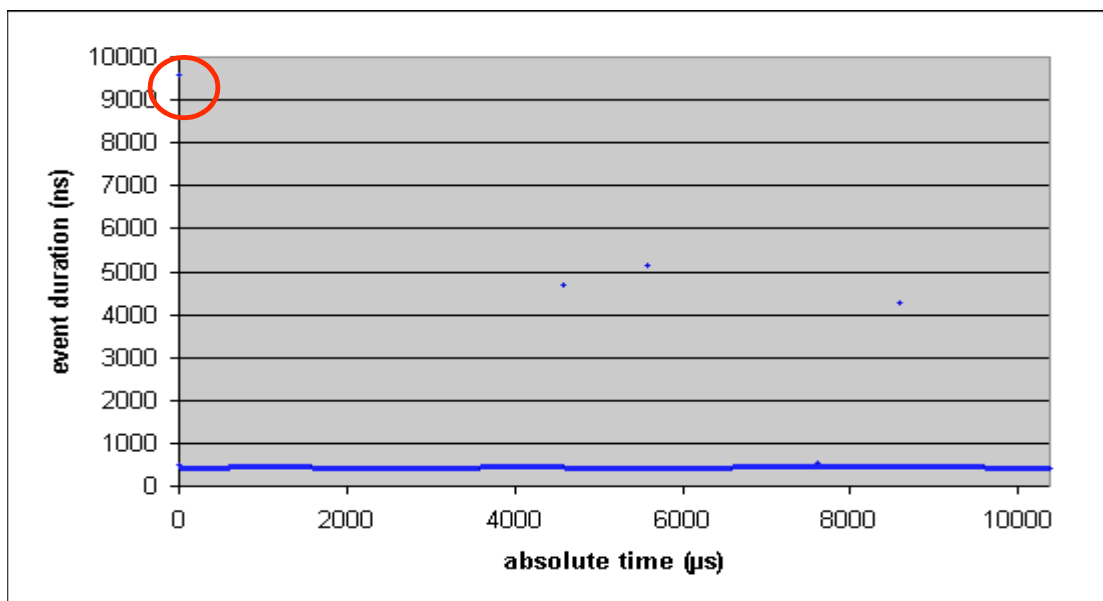


Figure 4.7-1 SEO-d-1.d

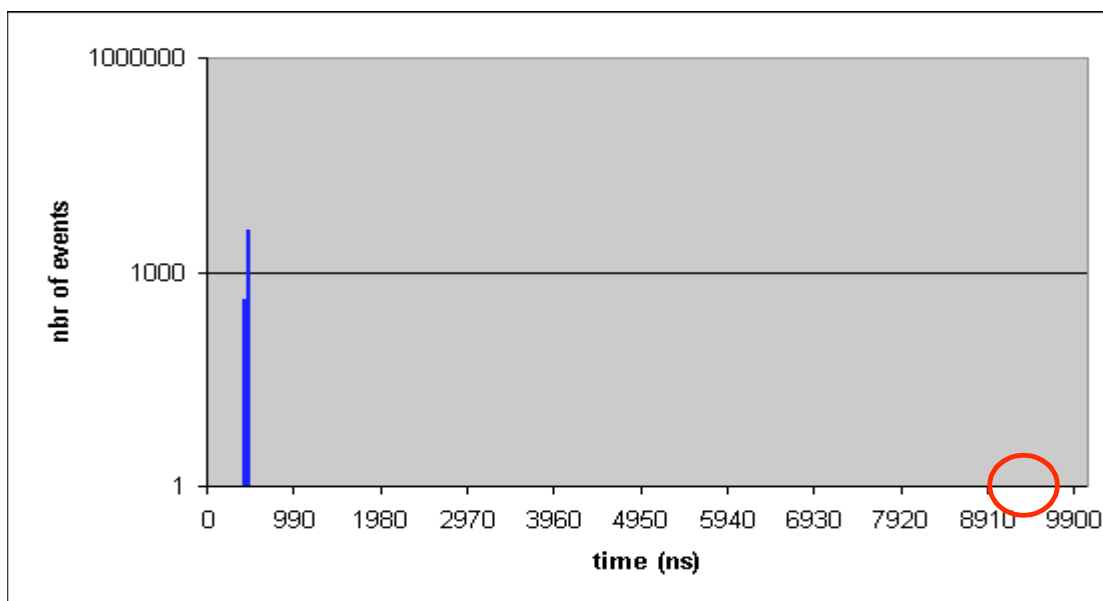


Figure 4.7-2 SEO-d-1.d - frequency distribution

4.7.2 No contention – Release (SEO-e-1.d)

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
8191	0.4	7.2	0.5

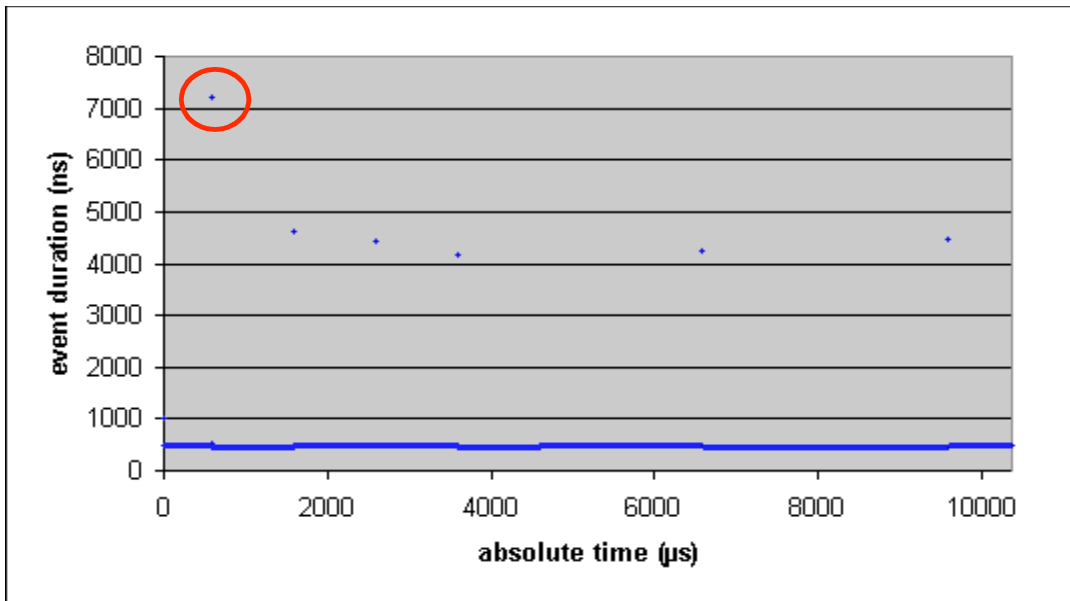


Figure 4.7-3 SEO-e-1.d

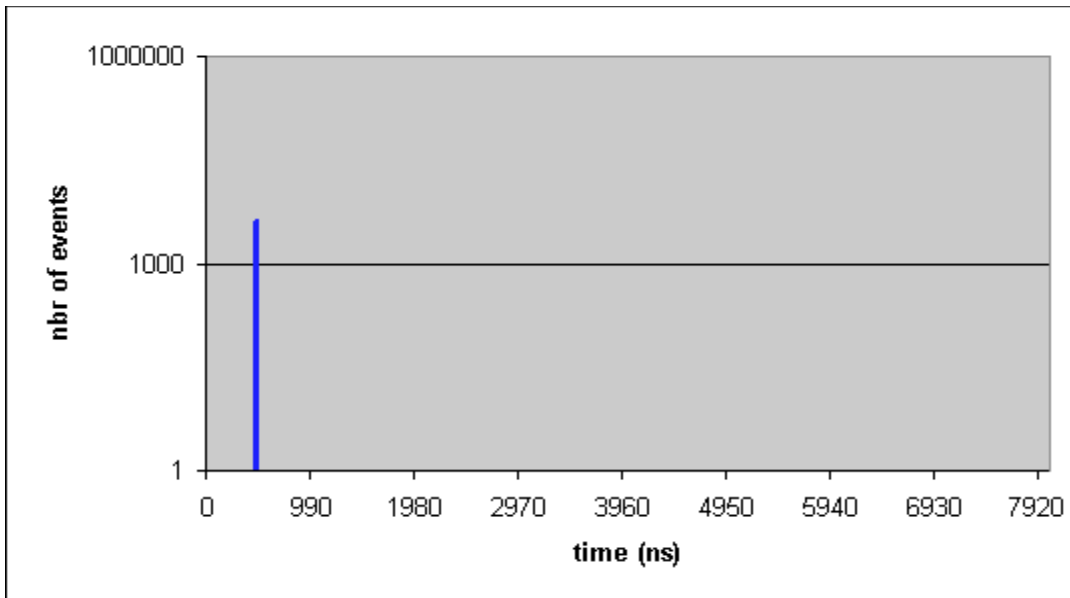


Figure 4.7-4 SEO-e-1.d - frequency distribution

4.7.3 Priority inversion (SEO-g-3.d)

- Number of threads: 3
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
10922	7.1	13.7	7.4

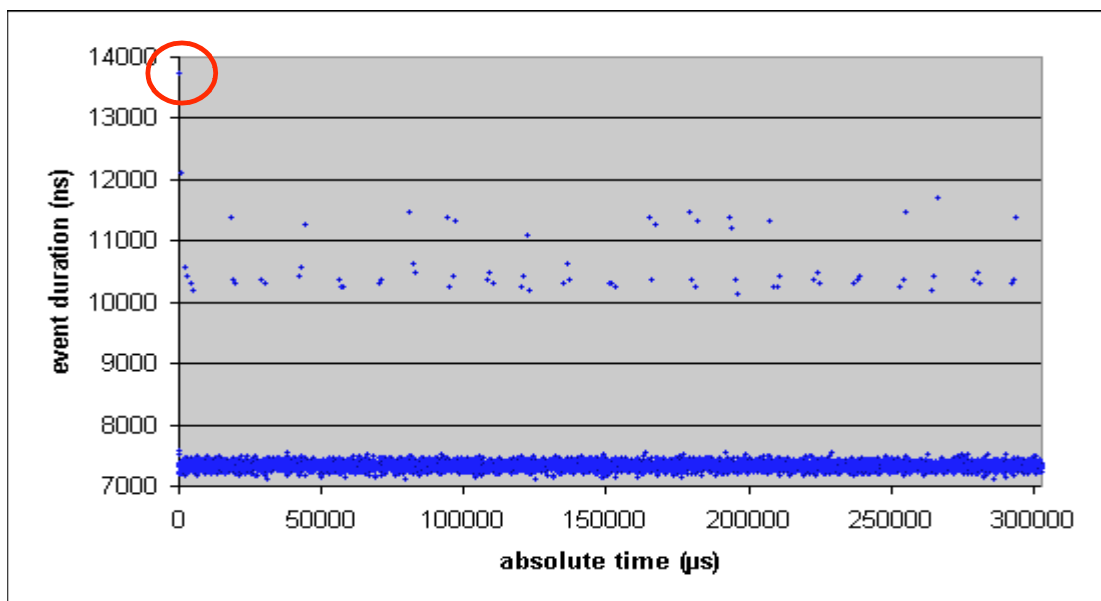


Figure 4.7-5 SEO-g-3.d

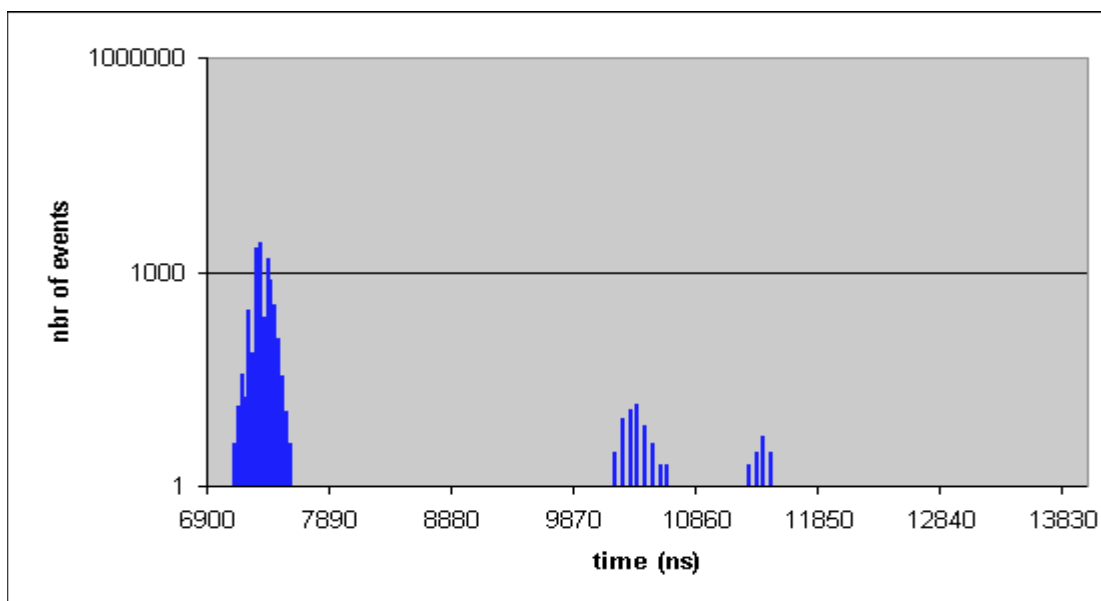


Figure 4.7-6 SEO-g-3.d - frequency distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.8 File system – Proprietary

The tests in this section evaluate the performance of the QNX file system.

We first measured the time it takes to create and delete a file. The average time to create a file was 5 ms, while the maximum was at 15 ms. This maximum occurred during the first sequence of the test.

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), 1 block of 512 bytes (1b) and 10 blocks of 5120 bytes (10b). We used a synchronous read/write operation so that we had an indication of the total time to read or write the data. Each reading and writing is completed only when the data has been transferred to the storage device and all file system information of the I/O operation is updated.

Real-time file systems are usually very simple file systems. Files need to be contiguous, so there are no unexpected head movements of the disk to other sectors or cylinders, which adds unpredictability to the system.

The QNX Neutrino RTOS v6.2 supports a range of file systems, from simple to complex. Its native file system, Fsys, supports directories, symbolic links and even file access rights for individual users or groups of users. Furthermore, the files are not guaranteed to be contiguous. Large files consist of multiple “extents”. An extent is a contiguous set of blocks on disk. However, the extents themselves that make up the file are not necessarily contiguous.

The fact that files are not contiguous is clearly demonstrated in section 0 (p. 77), which handles the read of 1 block (512 bytes). The average read operation takes about 150µs to complete, but periodically the same read operation takes four to five times as long.

The same conclusions can be drawn for the write operations. When using this file system in a real-time application, the designed needs to take this fragmentation into account, and has to make sure the time to continue with a new extent is bounded.

The QNX NEUTRINO RTOS v6.2 also supports a raw file system, which is basically a randomly accessible sequence of bytes that has no other predefined internal structure. It provides a more predictable response, but the applications are responsible for recognizing structure in this raw data.

4.8.1 Creation

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
250	6207	11596	6575

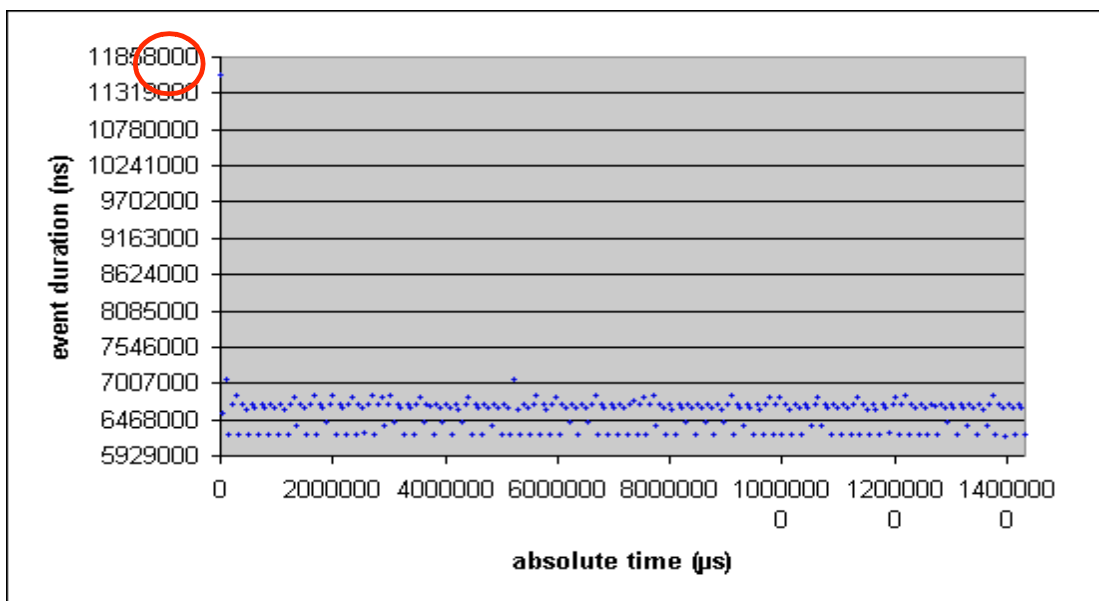


Figure 4.8-1 FS-a-1.d

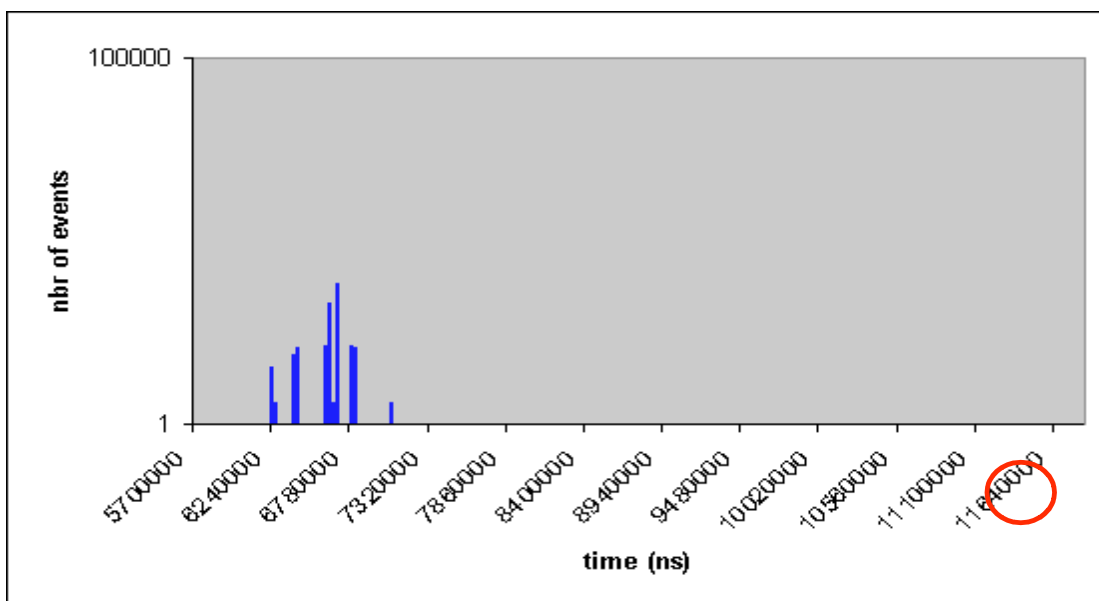


Figure 4.8-2 FS-a-1.d - frequency distribution

4.8.2 Deletion

- Number of threads: 1
- Memory model: d

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
250	561	759	578

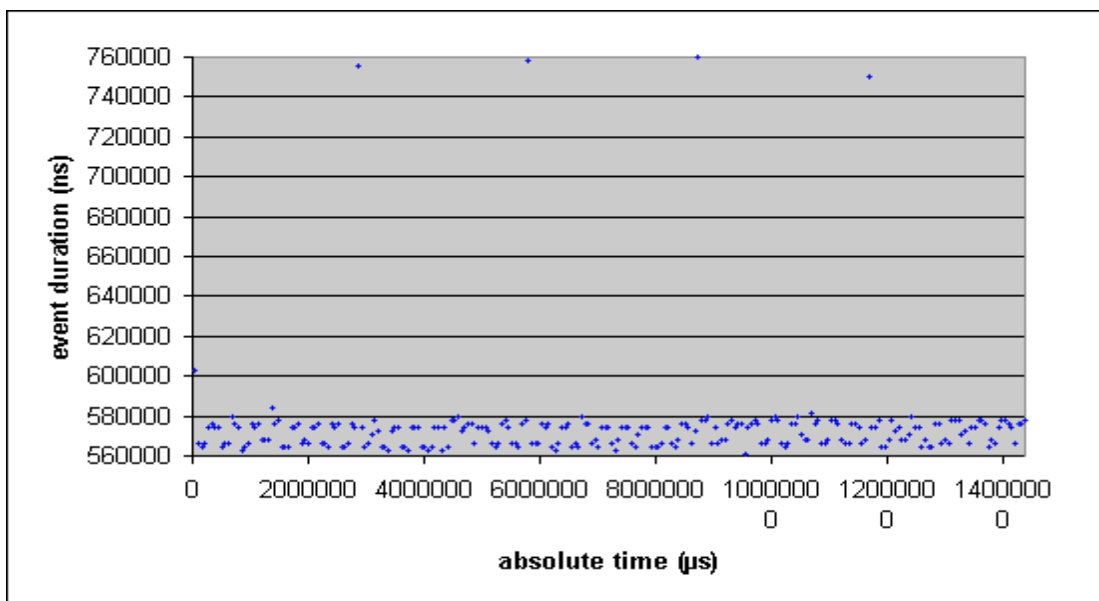


Figure 4.8-3 FS-b-1.d

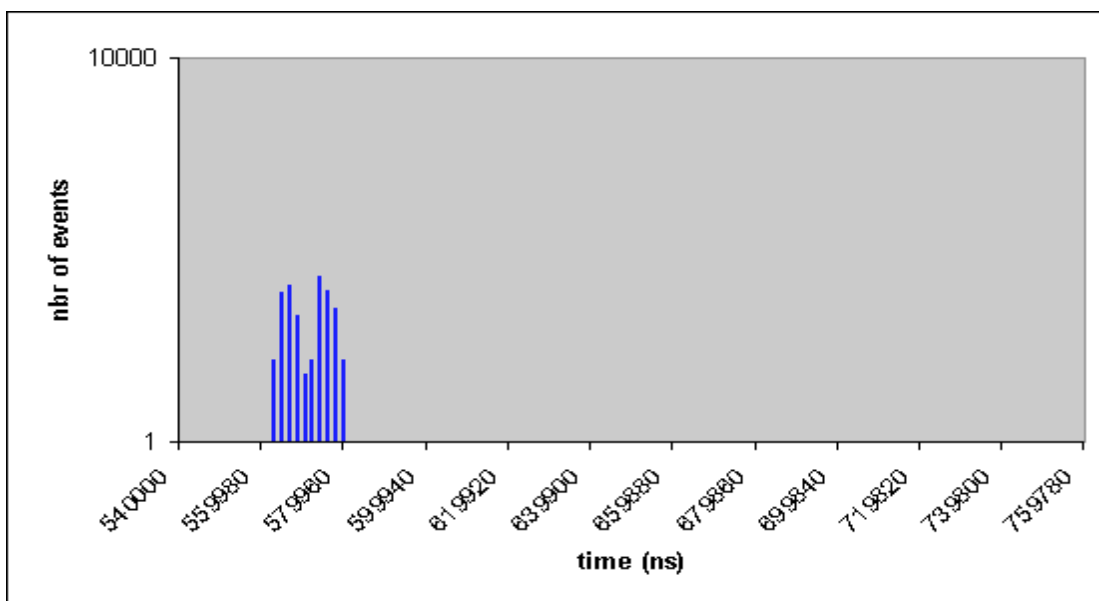


Figure 4.8-4 FS-b-1.d - frequency distribution

4.8.3 Read – 1 byte

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	91	116	93

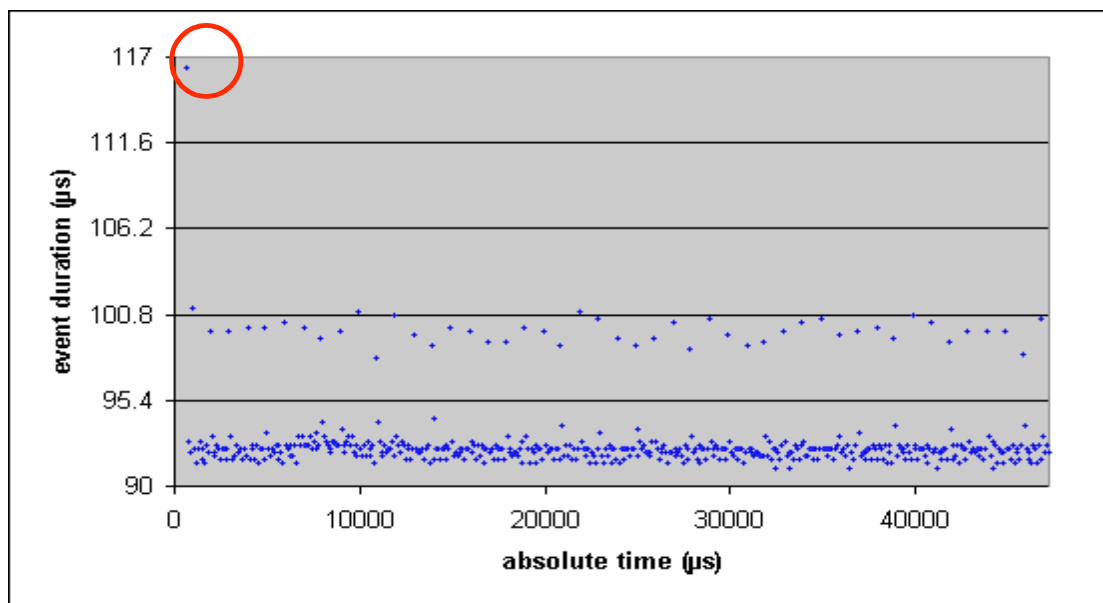


Figure 4.8-5 FS-c-1.d.1

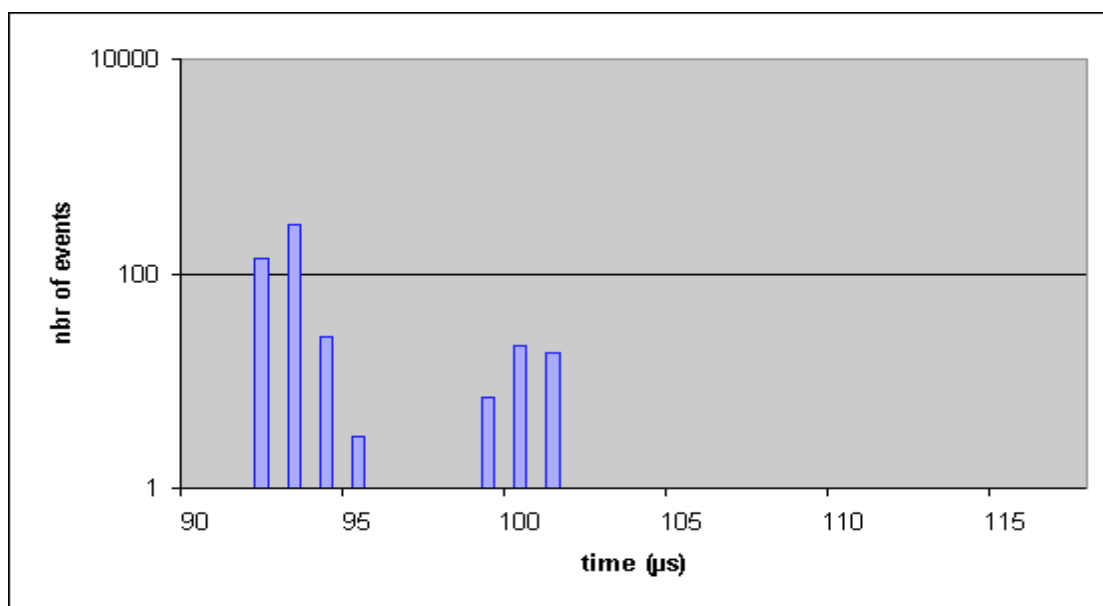


Figure 4.8-6 FS-c-1.d.1 - frequency distribution

Read – 512 bytes

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	114	440	123

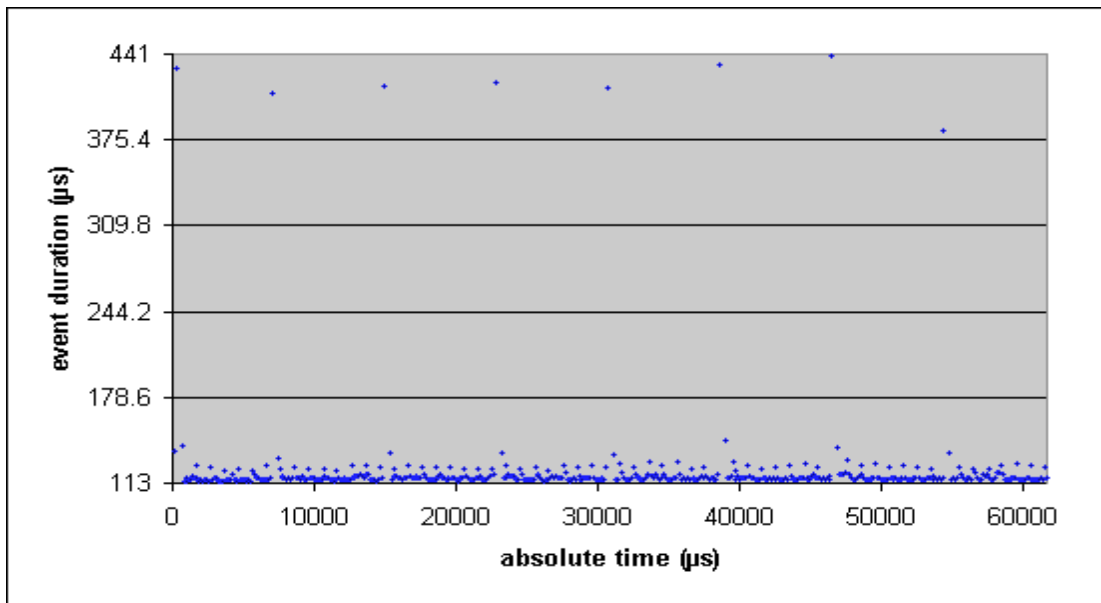


Figure 4.8-7 FS-c-1.d.1b

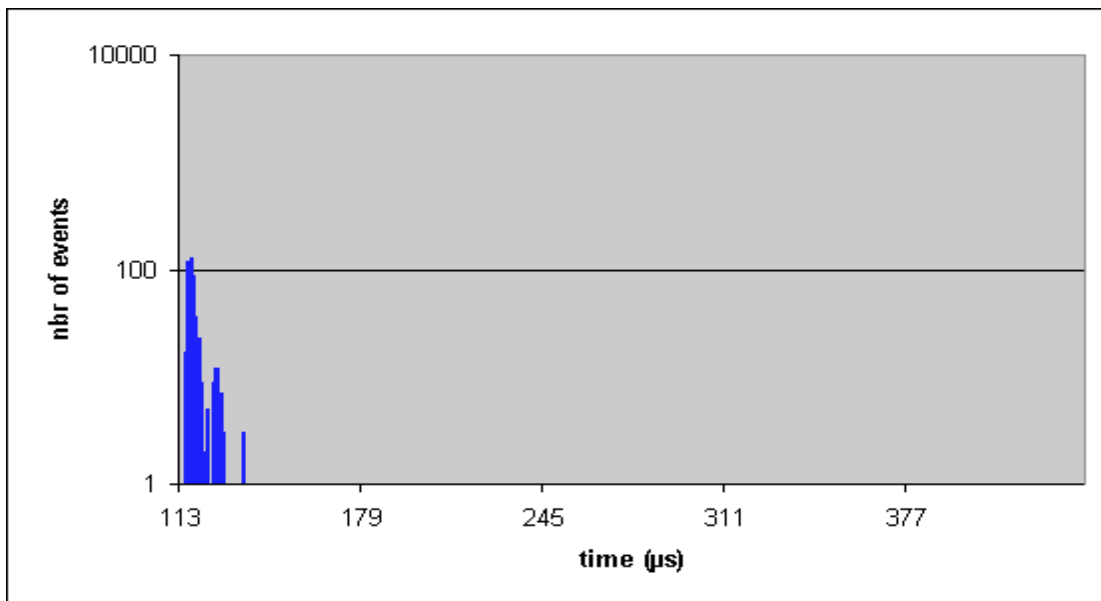


Figure 4.8-8 FS-c-1.d.1b – frequency distribution

4.8.4 Read – 5120 bytes

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	239	710	326

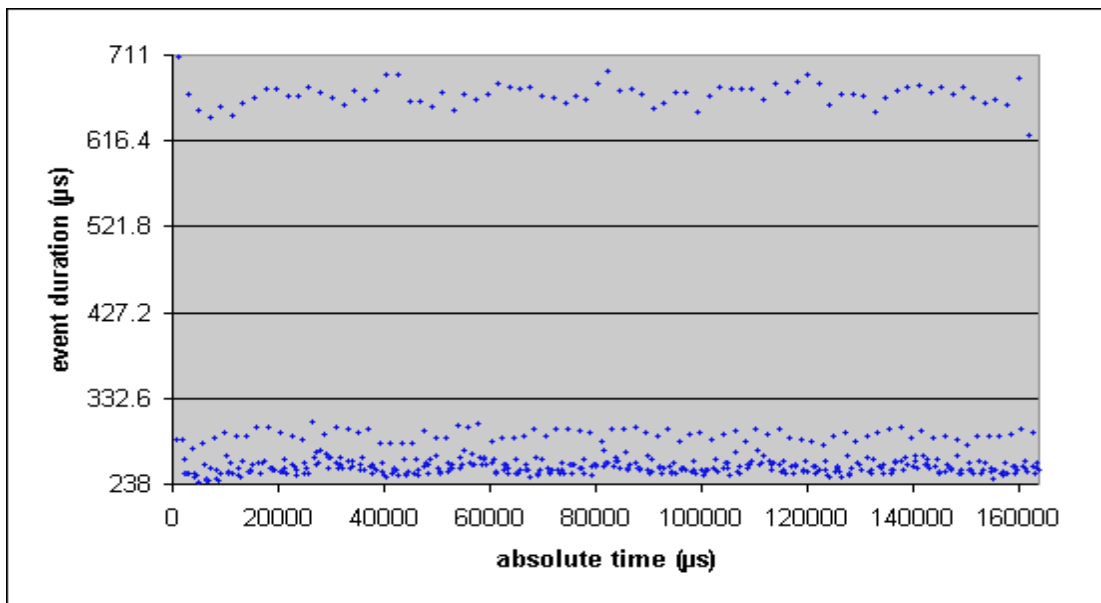


Figure 4.8-9 FS-c-1.d.10b

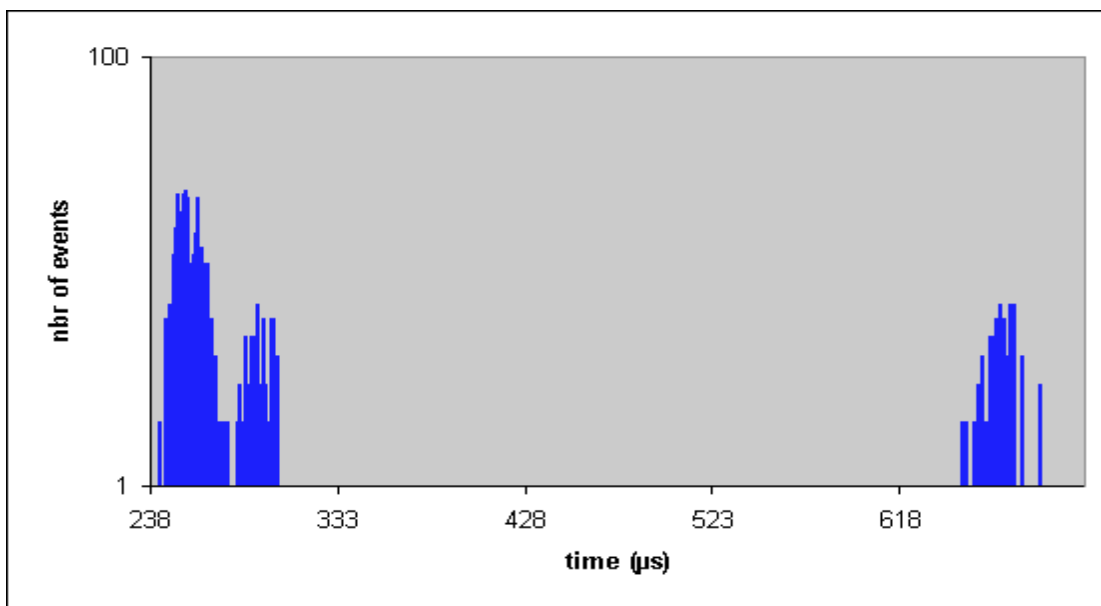


Figure 4.8-10 FS-c-1.d.10b - frequency distribution

4.8.5 Write – 1 byte

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	1710	29998	2340

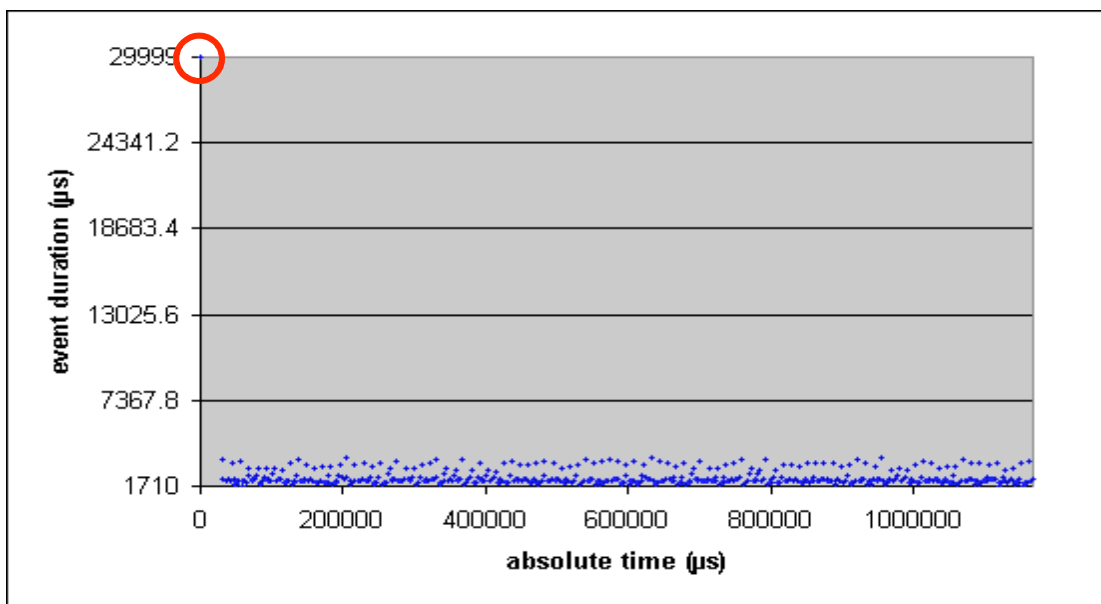


Figure 4.8-11 FS-d-1.d.1

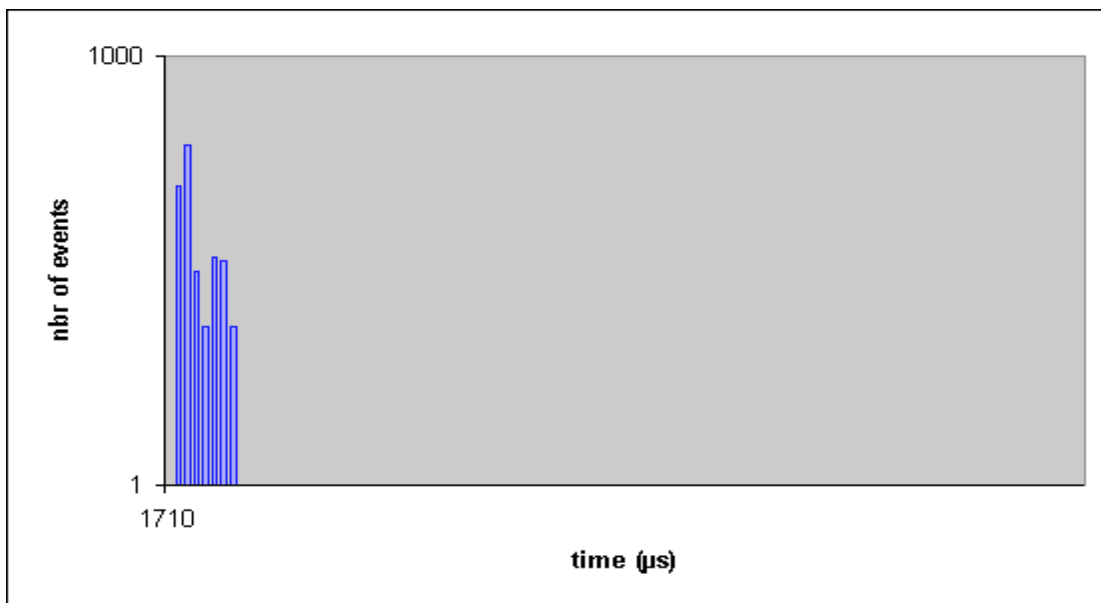


Figure 4.8-12 FS-d-1.d.1 - frequency distribution

4.8.6 Write – 512 bytes

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	30	54369	20936

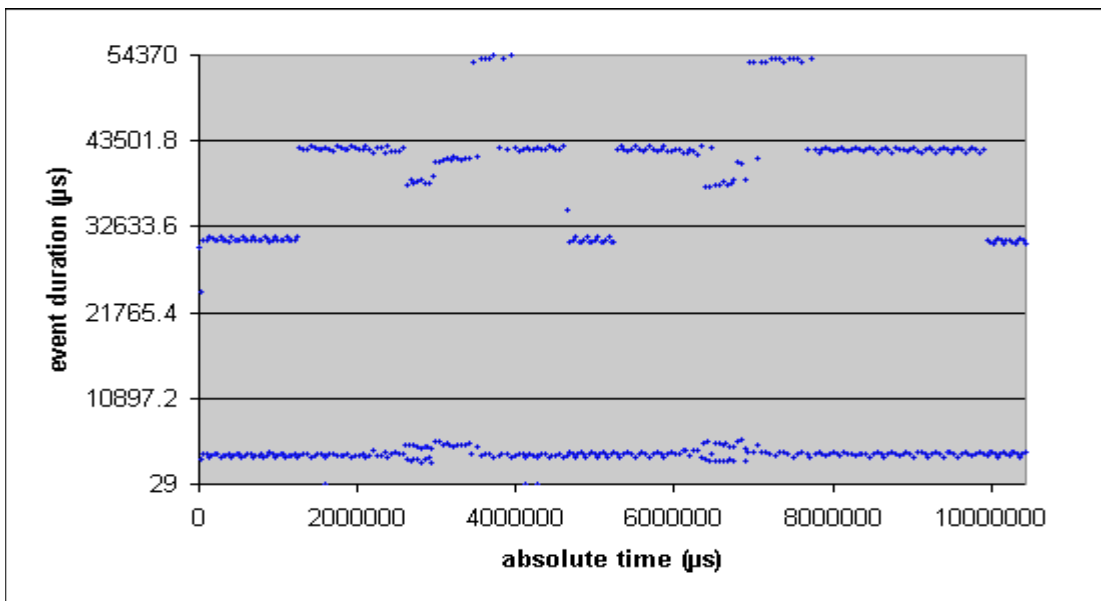


Figure 4.8-13 FS-d-1.d.1b

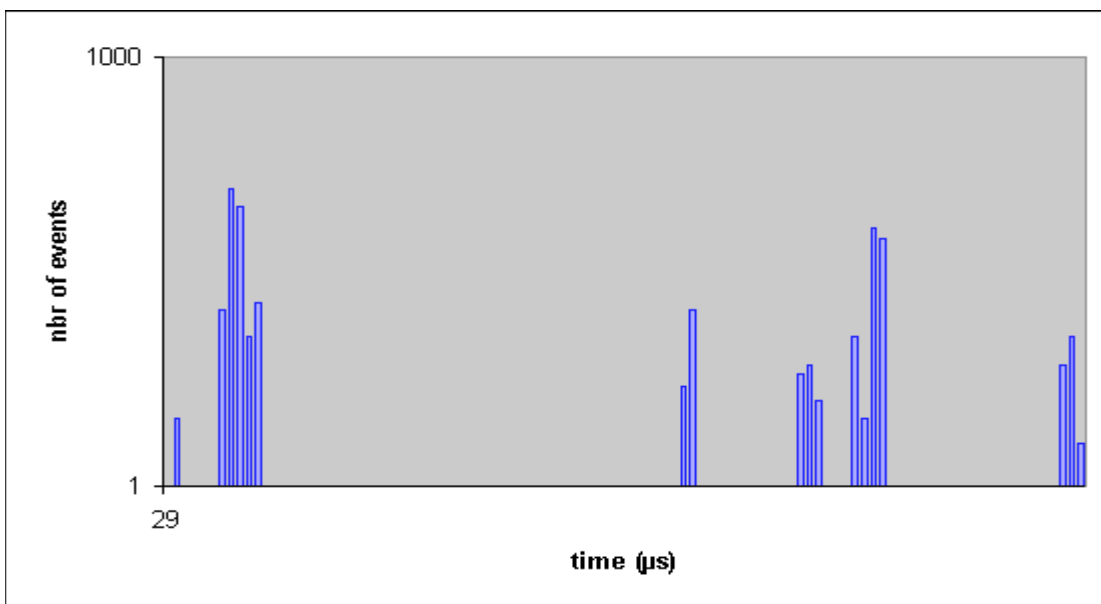


Figure 4.8-14 FS-d-1.d.1b – Frequency Distribution

4.8.7 Write – 5120 bytes

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

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
500	3911	58778	24519

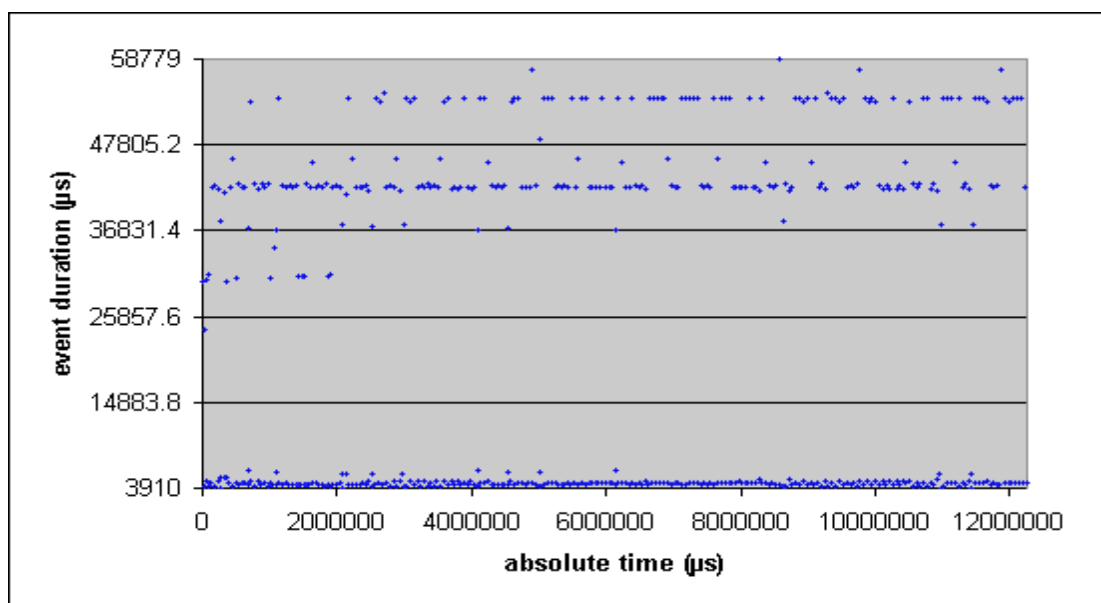


Figure 4.8-15 FS-d-1.d.10b

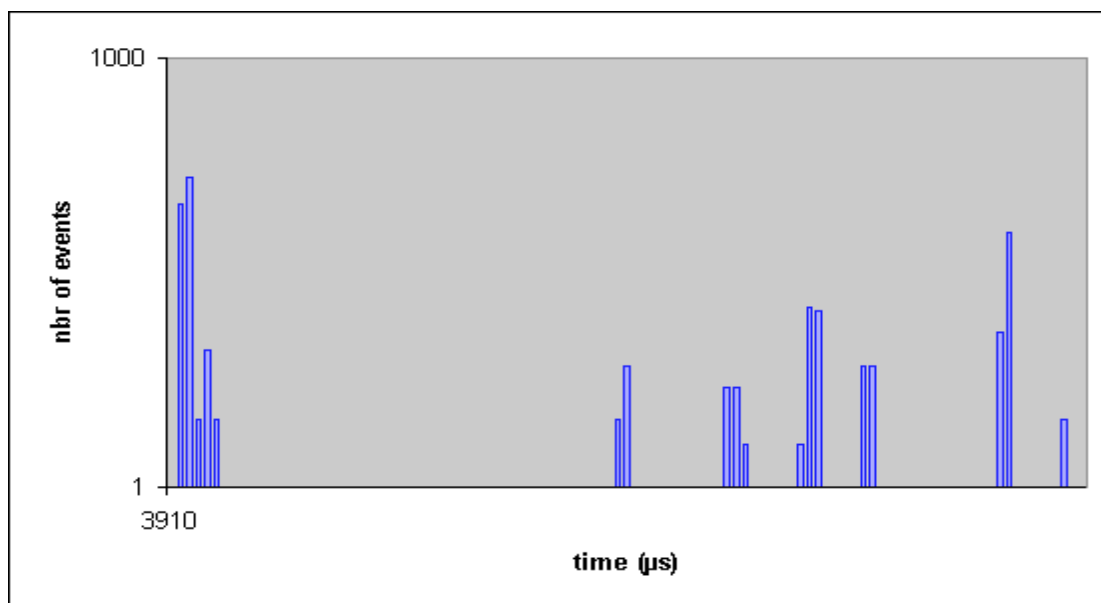


Figure 4.8-16 FS-d-1.d.10b – frequency distribution



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.9 Network stack – TCP/IP

In this section, we test the performance and efficiency of the TCP/IP network stack. The QNX NEUTRINO RTOS v6.2 provides two TCP/IP stack implementations:

- Full TCP/IP stack: a full implementation of the TCP/IP stack.
- Tiny TCP/IP stack: a tiny implementation of the TCP/IP stack for the use in resource-constrained environments.

Since our test application only uses basic socket calls, it was run on both stacks to compare their efficiency and performance.

The purpose of the test is to measure bandwidth and CPU usage for varying packet sizes. The packet size varies from 1 to 2000 bytes, in increments of 40 bytes. For more information concerning the implementation of this test, the reader is referred to the document “report definition and test plan”, which can be downloaded from <http://www.dedicated-systems.com/encyc/buyersguide/rtos/rtosmenu.htm>.

The tests were executed on an isolated 10 Mbit/s ethernet network. The sender disables the NAGLE algorithm. This algorithm can bundle several small data payloads into one bigger data payload, which is then sent in one TCP/IP packet. This mechanism reduces the protocol overhead significantly, but is often inappropriate in real-time systems because of the delay that is introduced.

4.9.1 Full TCP/IP Stack – Receive Capacity

Figure 4.9-1 shows that the receive capacity of the full TCP/IP stack approaches 1000 Kbytes/s when packet sizes increase. The stack used nearly 100% of the available CPU power, regardless the packet size.

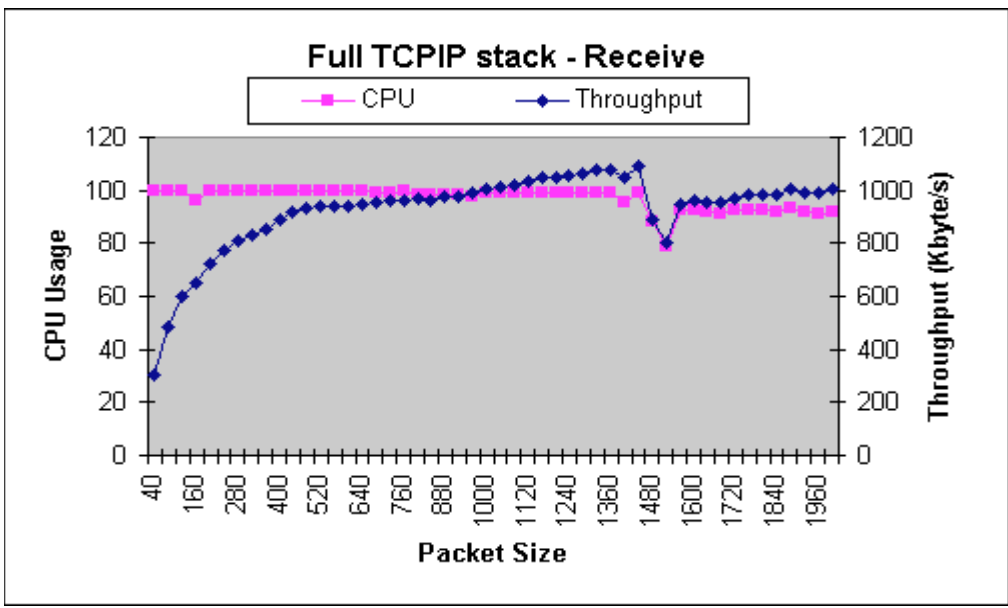


Figure 4.9-1: Full TCP/IP stack – receive capacity

4.9.2 Tiny TCP/IP Stack – Receive Capacity

The tiny TCP/IP stack's receive performance is a bit smaller than that of its full implementation counterpart, and uses less of the available CPU power.

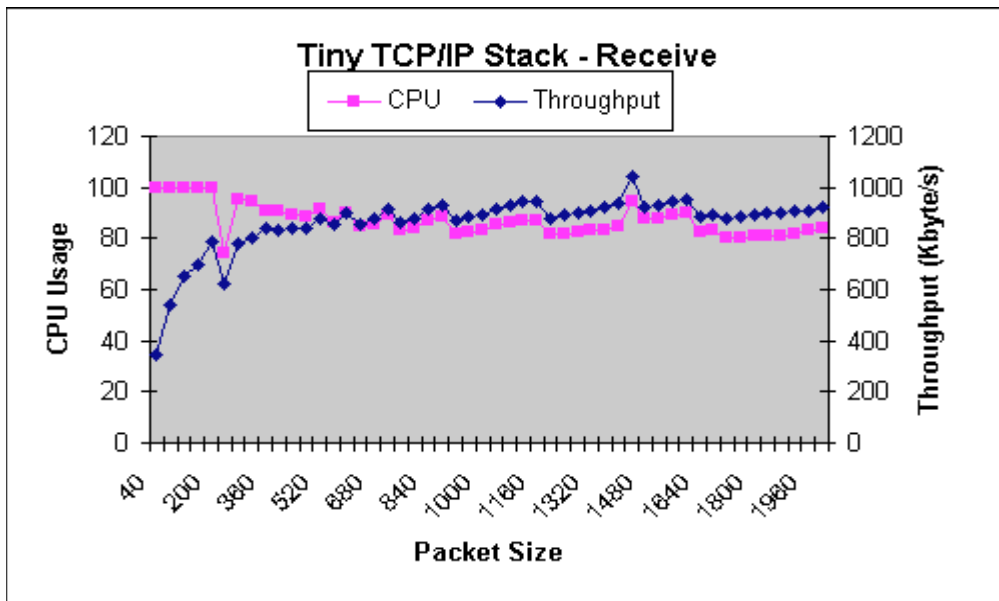


Figure 4.9-2: Tiny TCP/IP stack – receive capacity

4.9.3 Full TCP/IP Stack – Send Capacity

The same conclusions apply as the test in section 4.9.1 (Full TCP/IP Stack – Receive Capacity).

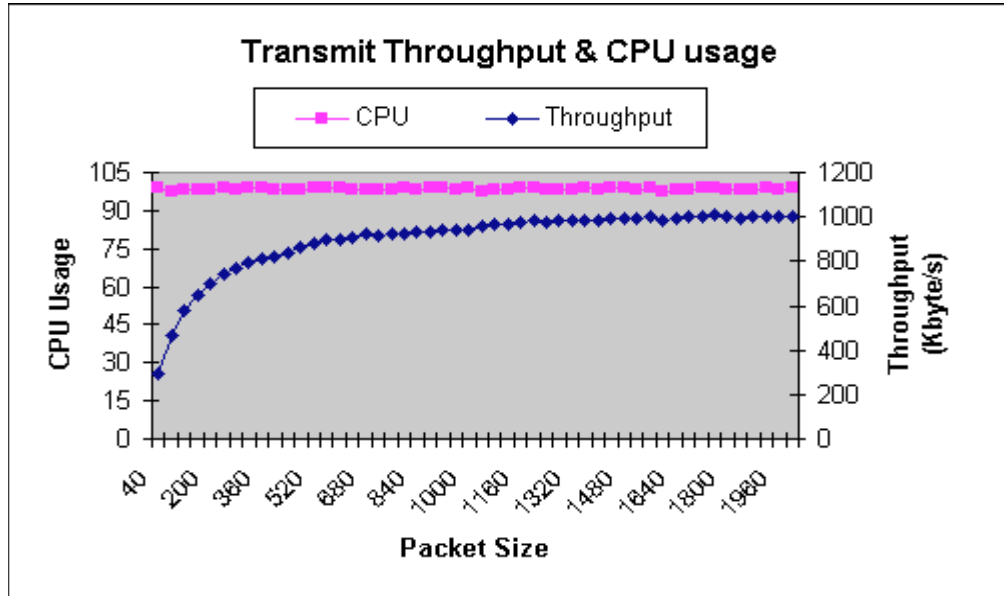


Figure 4.9-3: Full TCP/IP stack – send capacity

4.9.4 Tiny TCP/IP Stack – Send Capacity

The same conclusions apply as in section 4.9.2 (Tiny TCP/IP Stack – Receive Capacity). The send capacity of the tiny stack is significantly less than that of its full implementation counterpart. In general, the tiny TCP/IP stack's performance is much more dependent on the packet size, as can be seen Figure 4.9-4.

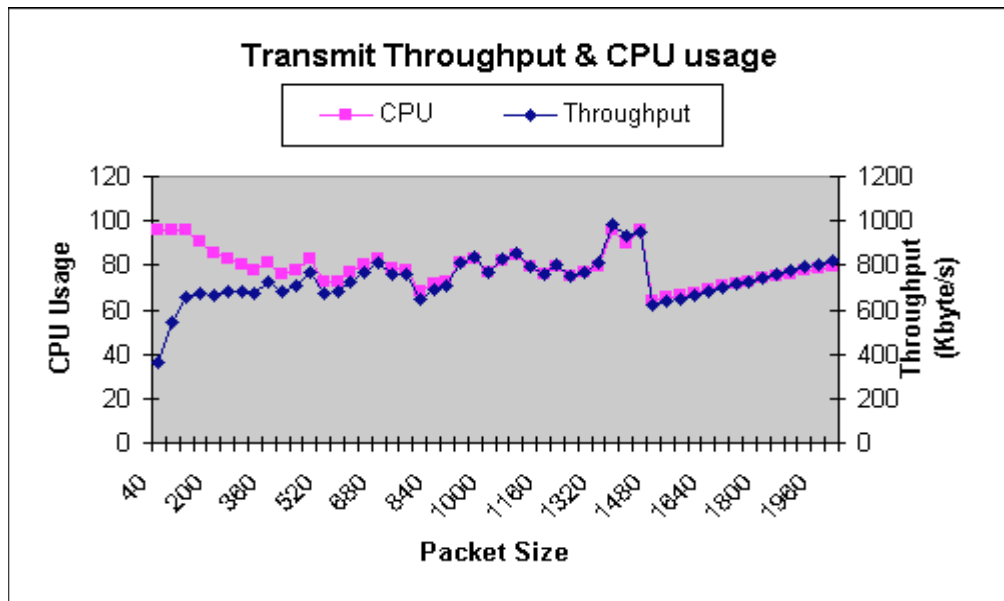


Figure 4.9-4: Tiny TCP/IP stack – send capacity



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.10 Clock Interrupt Service Routine

In this section, the clock ISR is discussed. The QNX NEUTRINO RTOS v6.2 is available for a 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 level.
- Targets where the clock interrupt does not have the highest priority level.

Ordinary Intel based PCs (by default) belong to the first category. The clock interrupt is hard-wired to IRQ0, which has a default 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.

Two 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 224 μ s to execute. When the RTOS clock interrupts the high priority main calculation thread, a longer calculation time will be measured. The difference between the normal (minimum) duration and worst case duration is the worst case clock interrupt duration (no other interrupts are active in the system).

The first test, for which the results are displayed in section 4.10.1, consists of the main thread only. As can be seen in Figure 4.10-1, an uninterrupted calculation takes 224 μ s to execute. The second band of samples located at 228 μ s is due to the clock interrupting the calculation. A conclusion that can be made is that a standard clock ISR takes about 4 μ s to execute (on a Pentium 200 MHz).

☺ This test was repeated with other threads added to it that execute "sleep" statements that expire in the middle of the main thread's calculations (this is similar to installing a timer). The results are not shown because the additional threads did not have an impact on the clock ISR execution time.

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 μ s 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 may need to be performed. In some systems, this causes the clock



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

ISR execution time to rise significantly. As mentioned before, this is a problem in targets where this clock interrupt has the highest priority, since it reduces the responsiveness of the system to other interrupts.

4.10.1 Clock ISR – no system load – no timers

- Number of threads: 1
- Memory model: a

Number of samples	Minimum (µs)	Maximum (µs)	Average (µs)
16383	223.9	230.6	224.8

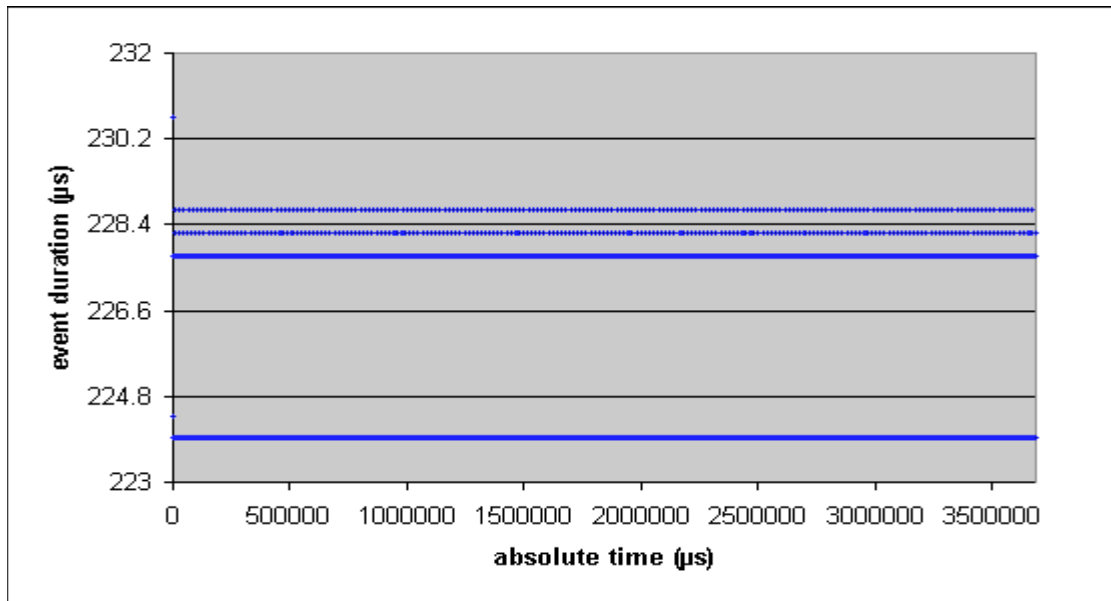


Figure 4.10-1 Clock ISR (no load, no timers)

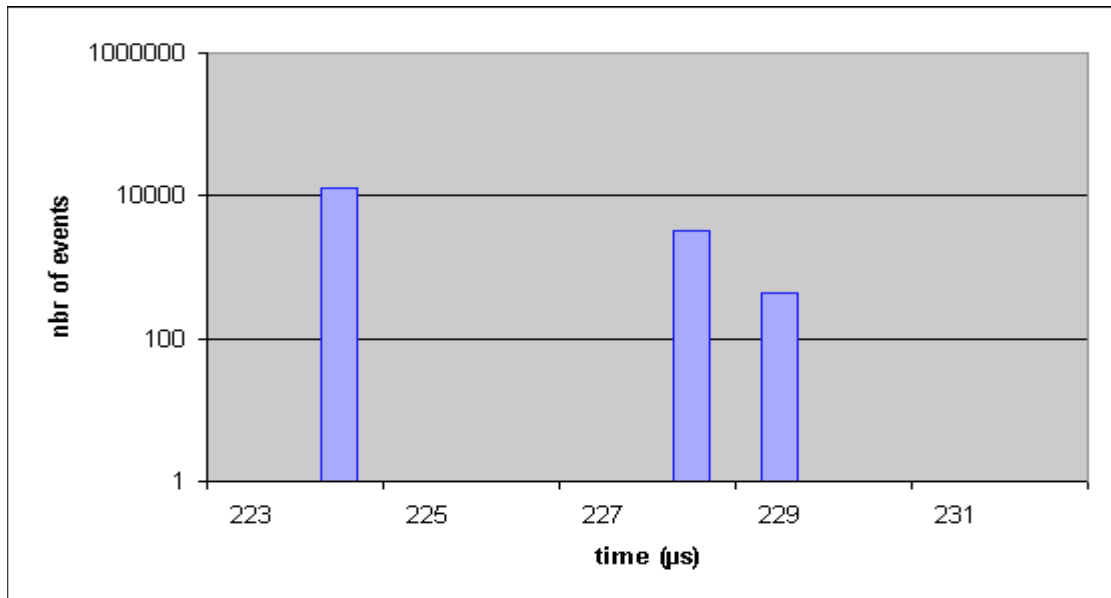


Figure 4.10-2 Clock ISR (no load, no timers) – Exploded View



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.11 Maximum number of objects

In this section, we verify whether or not creating a large amount of objects (semaphores and threads) impair the system's performance. This is accomplished by executing the thread switch latency test with 128 threads (see section 4.4) when the system is "loaded" with a large number of objects.

To verify if this large amount of objects has an impact on the system's performance, the test results are compared with those of the same test in "unloaded" condition.

The maximum number of objects is defined as the maximum number of objects that can be created without getting an "insufficient memory" error message during the execution of the test. Ideally, the results of the test executed with this maximum number of objects would be similar as the test results when the system is not loaded.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.11.1 Semaphores

The purpose of this test is to verify if creating a large number of semaphores has an impact on the system's performance. Therefore, the thread switch latency test with 128 threads will be executed when the system is "loaded" with semaphores.

We observed that the number of semaphores that can be created without running out of memory is astronomical in the QNX NEUTRINO RTOS v6.2. We decided to run the test with 100,000 semaphores. These semaphores serve no useful purpose whatsoever during the test, they simply present a load to the system.

The figure is shown together with the thread load in next section of this document.

4.11.2 Threads

The purpose of this test is to verify if creating a large number of threads has an impact on the system's performance. Therefore, the thread switch latency test with 128 threads will be executed when the system is "loaded" with 10,000 additional threads. Those additional threads serve no useful purpose whatsoever during the test, they simply present a load to the system.

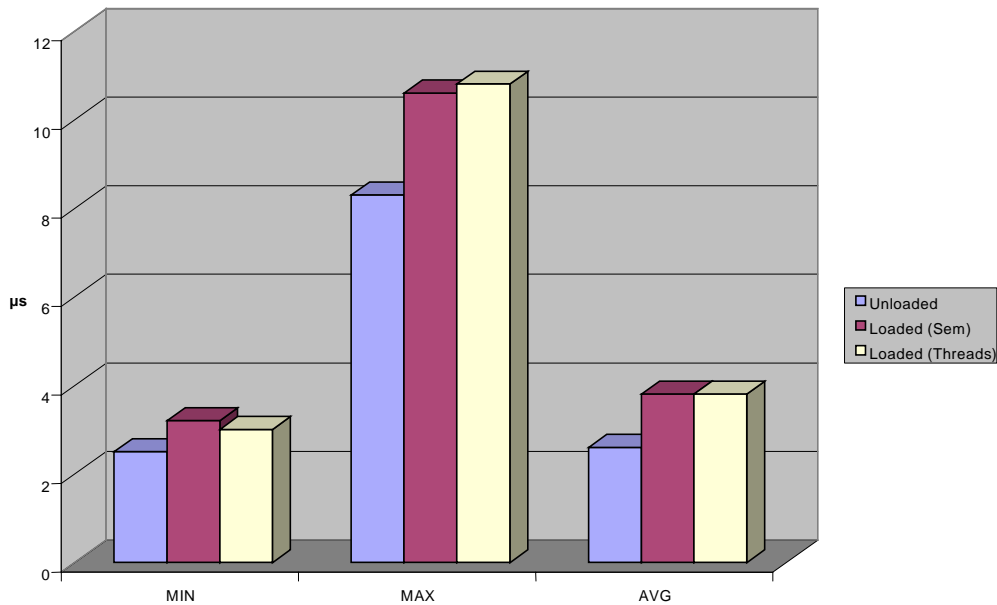


Figure 4.11-1 Thread Switch Latency Comparison – Loaded versus Unloaded

☺ As can be seen from Figure 4.11-1, the minimum, average and maximum latency increase somewhat in the loaded condition, but the maximum remains reasonable at 11µs. There is no gradual performance loss when the system's memory space is filled up with threads.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12 Memory Leaks

This section investigates if various system operations cause memory leaks.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.1 Semaphore and Mutex Creation/Deletion

The purpose of the tests in this section is to verify if semaphore/Mutex creation and deletion system calls cause memory leakage to occur.

The test is implemented as follows (pseudo code):

```
LOOP 1000 times
    Get Available Physical Memory
    Create 100 Objects
    Delete 100 Objects
END LOOP
```

☺ No memory leaks were detected.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.2 Process Creation/Deletion

In this section it is verified if creating and removing processes causes memory leakage.

The test consists of a parent process which forks a child process. Four different flavors of the test exist and are executed in the following sections.

- 1) The child process does not perform any work, allocate memory or creates objects and gets the chance to finish execution and exit properly.
- 2) The child process creates a number of semaphore objects which are not deleted. It finishes execution and exits properly.
- 3) The child process does not perform any work, allocate memory or creates objects but does NOT get the chance to exit properly. Instead the parent process unconditionally terminates it during its execution.
- 4) The parent process unconditionally terminates the child process during its execution, but not before it has created a number of semaphore objects that are not deleted.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.2.1 Normal Process Exit – No Object Creation

This test is very similar to the one executed in the thread test, but with processes instead of threads. The following test is executed (pseudo code).

```
LOOP 1000 times
    Get Available Physical Memory
    Create 1 PROCESS
    Sleep 10ms
    CloseHandle(PROCESS) // Close the handle to the process
END LOOP
```

The main loop creates another process and sleeps 10ms to let the process execute and exit. The created process contains only one thread which does not perform any work, it just writes one trace to the PCI analyzer so it can be verified that it has finished execution. Finally, the handle to the process is closed and the sequence starts over again.

☺ No memory leaks were detected.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.2.2 Normal Process Exit – With Object Creation

This test is identical to section 4.12.2.1 (Normal Process Exit – No Object Creation), except that the created process creates 100 semaphore objects that are not deleted when the process exits. By comparing the results with those of the previous section, conclusions can be made as to what happens with open object handles when a process exits.

☺ No memory leaks were detected.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.2.3 Forced Termination Process – No Object Creation

This test is similar to the one in section 4.12.2.1 (Normal Process Exit – No Object Creation), except that in this case the created process does not get a chance to finish executing and exit properly. The parent process kills the child during its execution.

☺ No memory leaks were detected.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

4.12.2.4 Forced Termination Process – With Object Creation

This test is similar to the one in section 4.12.2.2 (Normal Process Exit – With Object Creation), except that in this case the created process does not get a chance to finish executing and exit properly. The parent process kills the child during its execution.

☺ No memory leaks were detected.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

5 License policy and pricing

A non-commercial, x86 version of the QNX NEUTRINO RTOS v6.2 is freely downloadable from the vendor's website (www.qnx.com). The free download does not include the QNX Momentics IDE. Evaluation copies of the full QNX Momentics suite are available from QNX sales representatives. For production, QNX Software charges run-time license fees. Please contact the vendor for up-to-date pricing information.

6 Conclusions

☺ The QNX NEUTRINO RTOS v6.2 has a client-server architecture. Every process, including a device driver, has its own virtual memory space. The system can be seamlessly distributed over several nodes, and is network transparent.

☺ The system's performance is fast and predictable. None of the tests that were executed during this evaluation revealed any problems regarding performance, predictability or robustness.

☺ As opposed to the earlier versions of the QNX RTOS, which only supported the Intel x86 family of processors, version 6.2 also supports MIPS, PowerPC, ARM, StrongARM, XScale, and SH4.

☺ The new graphical System Builder Tool is a major improvement to configure your target. The dependencies checker works well and it has even a "diet" tool to shrink the shared libraries by removing unused functions.

☹ The documentation is extensive and improved against the previous version. It still lacks a good search tool.

☹ The Momentics IDE is an enhancement for the development tools, however you will need a decent host platform to get it running at a reasonable speed.

6.1 Comparison with QNX 6.1 (previous version)

6.1.1 Ratings

- Installation and Configuration rating increased from 7 to 8 due to the new graphical System Builder Tool.
- Tools rating increased from 7 to 8, due to the Momentics IDE
- Documentation and support increased from 5 to 7, as the arguments are now documented in the API

6.1.2 Test Results

- A small decrease in interrupt performance, but nested and simultaneous interrupts are handled a bit faster.
- Thread switch latency increased about 10%
- A small increase (less than 10%) in semaphore handling, but acquiring a non-signaled semaphore (which had a bad worst case time in QNX 6.1) improved a lot.
- The priority inversion case (priority inheriting mutex) improved a lot (30% decrease in worst case timing).
- As mentioned in the release notes, the ATA hard disk driver improved in performance (bloc DMA mode and ATA/66 support). This is clearly proven in our tests, showing an average increase in performance of 30%.
- As mentioned in the release notes the TPC/IP stack is improved. Our tests show indeed that there is an improvement, but it is small. This is normal, as the TCP/IP performance of QNX 6.1 was already very good.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

Our final conclusion is that in general there is a small decrease in performance, but a large increase of performance in some worst case scenarios. So the general real-time performance (where worst case behaviour is important) improved compared with QNX 6.1.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

7 Appendix A: Commentary from the vendor

Visit www.qnx.com to learn more about QNX Software Systems or to download the non-commercial version of QNX Neutrino v6.2.



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

8 Appendix B: Abbreviations

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



RTOS EVALUATION PROGRAM

Doc. Name: **QNX® Neutrino RTOS v6.2**

Doc. Version: **2.50** Doc. date: **13 August, 2002**

9 Appendix C: Document revision history

9.1 Issue 2.50 (August 13, 2002)

Initial issue.

Remark that the main version number increases if tests are added to our test suite. The extended test suite (Version 2) contains also network, disks and memory leak tests.