QNX NEUTRINO 6.5

Authors: Luc Perneel (1, 2), Hasan Fayyad-Kazan (2) and Martin Timmerman (1, 2, 3)

1: Dedicated Systems Experts, 2: VUB-Brussels, 3: RMA-Brussels

Disclaimer

Although all care has been taken to obtain correct information and accurate test results, Dedicated Systems Experts, VUB-Brussels, RMA-Brussels and the authors 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 these organisations and authors have been advised of the possibility of such damages.

http://www.dedicated-systems.com

E-mail: info@dedicated-systems.com

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

EVALUATION REPORT LICENSE

This is a legal agreement between you (the downloader of this document) and/or your company and the company DEDICATED SYSTEMS EXPERTS NV, Diepenbeemd 5, B-1650 Beersel, 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 CANNOT**:
   - You cannot, 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 cannot, 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.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

DOCUMENT CHANGE LOG

Issue No.	Revised Issue Date	Para's / Pages Affected	Reason for Change
1.00	25 March 2011	All	Initial draft
2.00	11 May 2011	All	Some textual improvements, extra sections on debug facilities after vendor draft revision
3.00	20 May 2011	All	Some textual improvements
4.00	27 May 2011	All	Some textual improvements
4.01	27 May 2011		Final draft
4.10	8 Sept 2011	All	Ready-to-go

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

1 Document Intention

1.1 Purpose and scope

This document presents the qualitative evaluation results of the QNX 6.5 operating system. The testing results of this operating system employed on various platforms can be found on our website. (www.dedicated-systems.com)[Doc. 5, 6 & 7].

The layout and the content of this report follow the one depicted in “The evaluation test report definition” [Doc. 3] and “The OS evaluation template” [Doc. 4]. See section 1.4 of this document for more detailed references. Therefore these documents have to be seen as an integral part of this report!

Due to the tightly coupling between these documents, the framework version of “The evaluation test report definition” has to match the framework version of this evaluation report (which is 2.9). More information about the documents and tests versions together with their corresponding relation can be found in “The evaluation framework” see [Doc. 1] in section 1.4 of this document.

1.2 Document issue: the 2.9 framework

This document shows the results in the scope of the evaluation framework 2.9.

1.3 Conventions

Throughout this document, we use certain typographical conventions to distinguish technical terms. Our used conventions are the following:

- ***Bold Italic*** for OS Objects
- **Bold** for Libraries, packets, directories, software, OSs...
- `Courier New` for system calls (APIs...)

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

1.4 Related documents

These are documents that are closely related to this document. They can all be downloaded using following link:
http://www.dedicated-systems.com/encyc/buyersguide/rtos/evaluations

Doc. 1 The evaluation framework
This document presents the evaluation framework. It also indicates which documents are available, and how their name giving, numbering and versioning are related. This document is the base document of the evaluation framework.
EVA-2.9-GEN-01 Issue: 1 Date: April 19, 2004

Doc. 2 What is a good RTOS?
This document presents the criteria that Dedicated Systems Experts use to give an operating system the label “Real-Time”. The evaluation tests are based upon the criteria defined in this document.
EVA-2.9-GEN-02

Doc. 3 The evaluation test report definition
This document presents the different tests issued in this report together with the flowcharts and the generic pseudo code for each test. Test labels are all defined in this document.
EVA-2.9-GEN-03 Issue: 1 Date: April 19, 2004

Doc. 4 The OS evaluation template
This document presents the layout used for all reports in a certain framework.
EVA-2.9-GEN-04 Issue: 1 Date: April 19, 2004

Doc. 5 QNX 6.5 on an X86 (Atom) platform
This document presents the layout used for all reports in a certain framework.
EVA-2.9-OS-QNX……. Issue: 1 Date: TBD

Doc. 6 QNX 6.5 on an ARM platform.
This document presents the layout used for all reports in a certain framework.
EVA-2.9-OS-QNX….. Issue: 1 Date: TBD

Doc. 7 QNX 6.5 on a PPC platform.
This document presents the layout used for all reports in a certain framework.
EVA-2.9-OS-QNX…… Issue: 1 Date: TBD

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

2 Introduction

2.1 Overview

QNX Software Systems Ltd was founded in 1980 and has been always focused on delivering solutions for the embedded systems market.

One of the main differences between **QNX** and other RTOS is the fact that **QNX** is built around the **POSIX** API standard. This has its advantages as a lot of code for **Linux** based platforms can be compiled and run on **QNX Neutrino**. However, bear in mind that we are discussing a real-time operating system here.

QNX Neutrino is based on true microkernel architecture with message-based inter-process communication. For instance, drivers are just applications with special privileges, and as such they cannot crash the kernel. The concept of kernel modules which is the case in **Linux** is not needed here, which makes **QNX Neutrino** a very stable product.

Furthermore, **QNX Neutrino** was initially built-up as a multi-processor capable operating system (both SMP and AMP). Nowadays, this is a very important asset in today's multi- and many-core business.

2.2 Evaluated (RTOS) product

The operating system that we are going to evaluate is the **QNX NEUTRINO RTOS v6.5.0**, from **QNX** Software Systems Ltd.

2.3 Supported CPU

QNX Neutrino v6.5.0 supports the following processors:

- x86
- ARM
- POWER architecture (PowerPC)
- MIPS
- SH-4

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

3 Evaluation results summary

Following is a summary of the results of evaluating the **QNX NEUTRINO RTOS v6.5.0**, from **QNX** Software Systems Ltd.

3.1 Positive points

- Excellent architecture for a robust and distributed system.
- Very fast and predictable performance.
- Large number of board support packages (BSP) and drivers (the source code for most of them was available for public) which can be easily downloaded.
- One of the best documented RTOS on the market.
- Efficient and user friendly Integrated Development Environment (IDE)
-

3.2 Negative points

- Not all the source code is available. Customers can apply for source access.
-

3.3 Ratings

For a description of the ratings, see [Doc. 3].

RTOS Architecture	0	9	10
OS Documentation	0	9	10
OS Configuration	0	8	10
Internet Components	0	8	10
Development Tools	0	9	10
BSPs	0	8	10
Support	0	8	10

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4 Technical evaluation

This document is limited to the technical evaluation aspects of the product and a qualitative approach is used. The quantitative test results on a specific platform (X86, Atom, ARM & PPC) can be found in other reports.

The scores in this document are given in respect of the experiences gathered from evaluating the product and its comparison with other competing products that we already tested or still under testing. Although all effort has been done to be as rigorous as possible in the scoring, one should accept that they are partially based on subjective criteria. Therefore, this part of the evaluation has as sole intention to help the application designer in his work. Indeed, each (commercial) RTOS is using particular architectural approaches and working models. This has a serious performance and behaviour consequences influencing considerably application designs.

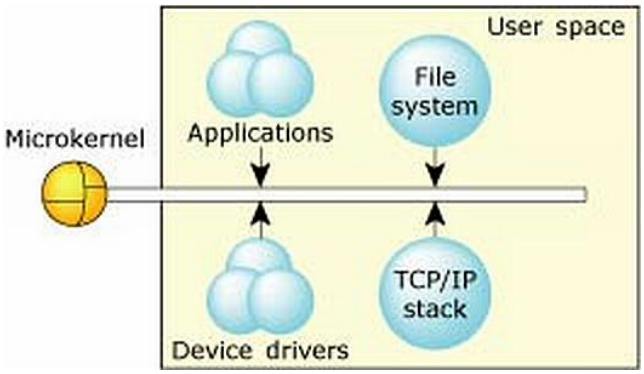
4.1 OS Architecture

RTOS Architecture 0 | | | | | | | | | 9 | | 10

*The **QNX NEUTRINO RTOS v6.5** has true client-server architecture, providing full virtual memory protection. It is a message based OS, and can seamlessly be distributed over multiple nodes. The RTOS supports SMP, and implements several HA (High Availability) features.*

The **QNX NEUTRINO RTOS v6.5** has a client-server architecture consisting of a microkernel and optional cooperating processes. The microkernel implements only the core services like threads, processes, signals, message passing, synchronization, scheduling and timer services. The microkernel itself is never scheduled. Its code is executed only as a result of: 1) a kernel call, 2) the occurrence of a hardware interrupt, 3) or a processor exception. The microkernel is not used for any call that would take a long execution path (or would be blocking), so that the longest non pre-emptable path in the kernel is bound.

Any additional functionality is implemented in the cooperative processes, which act as server processes and respond to the requests 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. This is shown in the figure below.



Microkernel architecture

While the kernel runs on the CPU at privilege level, all other processes run at user level. As a consequence, the managers and device drivers also run at user level. However, these processes can request the OS to give I/O privileges for their threads in order to access I/O and install interrupt handlers (which can be done only if the application process runs at root user). Depending on the used CPU type, the system call for giving I/O privileges for the thread can change the CPU privilege level of the running thread.

Application processes that do not request I/O privilege are always running at user level.

☺ The fact that device drivers and server processes are not part of the kernel, helps in debugging them easily (just like any other application) and so the kernel will not crash due to faults in any device driver! This makes **QNX** to be a very safe and reliable RTOS.

The **QNX NEUTRINO RTOS v6.5** 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 in turn 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.

Synchronous IPC mechanism using un-queued messages is the base for system calls throughout **QNX**. For this purpose, servers will create “channels” where other threads can send to. The concept is that a client sends a message to a certain channel, not to a specific thread. Channels use a small integer identifier (like the FD file descriptors upon open). In this way, they can be directly used for file descriptors as well.

The channel uses a priority FIFO queue for all pending requests. The priorities are inherited from the client thread that initiated the call. As such, the server thread will always start first by handling the highest priority pending request message.

Of course, it is not because the messages are prioritized that priority inversion can’t happen. It is indeed possible that the server thread happens to runs on a lower priority than the client thread.

To deal with this issue, **QNX Neutrino** puts two mechanisms in place:

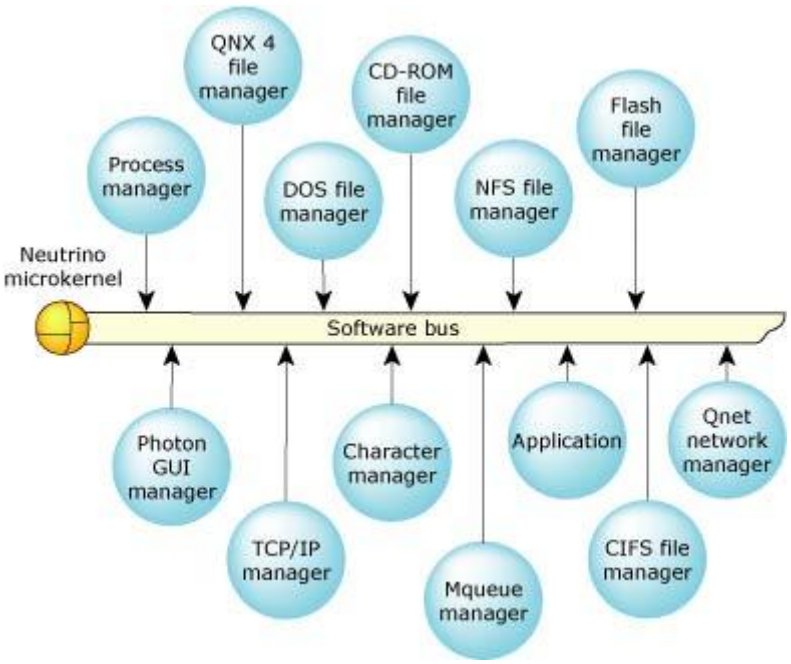
- When handling a "received" message whose priority is "lower" than the server priority, then the server priority will be lowered to the client thread priority. This prevents the lower priority client thread from indirectly blocking higher priority threads via the server.
- When a server thread is running with a priority which is lower than the "sending" client priority, then the server thread priority will be increased up to the same priority level as the sending client at the moment of the send (this means immediately).

☺ As a result, the relative priorities of the client threads requesting work of the server are preserved, and the server work will be executed at the appropriate priority. This message-driven priority inheritance avoids priority-inversion problems.

The use of this message-driven priority model is hidden behind classical procedural API's and therefore acts similarly as the concept of calling a procedure within the same thread. It runs automatically at the same priority as the caller, and does not induce any supplementary delay except if the message is passed across a LAN, where classical LAN delays will show up.

QNX Neutrino is fully pre-emptable, even during passing messages between processes; it resumes the message pass from the moment it left off before pre-emption.

QNX uses a software bus model. Software modules can be plugged-in on demand when extra features are needed. This is illustrated in the figure below.



The software bus concept.

☺ This client-server architecture has many advantages. Probably, the most important one is robustness. Every manager and device driver runs in its own virtual memory address space, resulting in a robust and reliable system. The only exception is the process manager itself as it requires the capability to manage all

processes. In theory, the assumption is that the price to pay is performance: execution of system calls results in a couple of context switches with an overhead caused by memory protection, which should theoretically result in somewhat lower performance. However, test results show that this performance trade-off is very low and even not observable. Client-server architecture makes coding simpler which should improve development time efficiency.

4.1.1 Task Handling Method

4.1.1.1 Processes and threads

The **QNX NEUTRINO RTOS v6.5** uses processes and threads. A process defines the address space in which threads will run, and will always contain at least one thread. This is the same view as the definition we adopted throughout all our reports. As a consequence, processes protect its collection of threads with regards to the other threads in the system.

The scheduler makes all scheduling decisions based on threads without any regards to process boundaries.

☺ Compared to our previous report on **QNX 6.3**, an interesting feature/attribute has been added: the thread name. This makes it easier when debugging/monitoring your application which in turn augments project design efficiency.

4.1.1.2 Thread priorities

QNX Neutrino supports 255 thread priorities which should be enough even for complex systems supporting a Rate Monotonic Scheduling model where it is mandatory to put threads on different priority levels.

Priorities are numerically ordered: for example, 0 is the lowest priority, and it is reserved for the idle thread.

The 255 user priority levels are subdivided by default into different sections:

- Priority 0: reserved for the idle thread
- Normal priorities 1 – 63: that may be used by any thread
- Privileged priorities 64 - 255: these priorities may only be used by root processes (root as used in **UNIX** like operating systems: for processes with user id 0).

Should it be required, the process manager has a start-up parameter which changes the boundary between normal and privileged processes.

The concept of non-privileged processes (which may not use the full priority range) is useful to allow third-parties to develop non real-time applications next to a real-time application. The real-time part runs in privileged mode and the non real-time part does not interrupt high priority threads, as it doesn't have the privilege to do so.

4.1.1.3 Scheduling mechanisms

Given that **QNX** is a real-time operating system, a priority based scheduling approach is used to schedule threads. For threads belonging to the same priority level, the following different scheduling options can be specified:

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

- ***FIFO***: the thread stays at the front of the queue until it voluntary relinquish control (mostly by calling blocking system calls).
- ***Round-Robin***: same as ***FIFO***, but an extra time-slicing mechanism is used here, which will put the thread at the end of the queue when the time-slice is passed. (Note that a time-slice is 4 times the clock period).
- ***Sporadic scheduling***: this is a type of adaptive scheduling where a thread has two priorities. 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 are adaptable for each thread.

Remark that ***FIFO*** is not always used: a server thread that's receiving a message will come in the front of the queue (as it acts on the callers behalf). The latter is necessary to avoid latencies in synchronous calls.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.1.1.4 Process management.

The management of processes and memory in a **QNX** system is handled by the process manager. The process manager is a special process, in the sense that the microkernel is included in it and as such they share the same address space. This is logical construction as both need to have access to all memory in all processes.

Nevertheless, the process manager is NOT running in privileged mode and still has to perform a privacy switch when invoking the microkernel! This means that the process manager behaves just like any other process in the system.

In the "**QNX Neutrino RTOS** System Architecture" manual, which is a very well written and understandable document, **QNX** proposes, for maximum robustness, to avoid using threads when there is no need for them and instead to use in such cases processes with a shared memory partition between them. It is not that they are against thread usage, but threads should only be used when they make sense (for example to introduce concurrency) and processes should be used for decoupling, modularity and protection (loosely coupling is always good in software systems).

They defend the use of processes through the following arguments:

- The context switch between processes is not much longer than between threads (the difference lays in the MMU/addressing context).
- If you use shared memory between processes, then you do not have the copy overhead.
- As a plus, you have protection mechanisms in place between the threads (as now they are processes).

Shared memory is protected from all processes except the ones where it is shared with. So although the shared memory itself is not (fully) protected (so one process can still corrupt shared memory used by another process), the local thread data (and thus thread stack) is protected from the other threads (as they are separate processes).

Of course, the decision about what to put in threads and what in processes is a fundamental design step which is not to be taken lightly. As usual, neither the solution of using one process full of threads, nor the solution of using only processes without threads will be optimal… It is however true that a lot of embedded software engineers do not take processes into account, as historically most RTOS did not have such protection concept (today this is still true for a lot of RTOS).

As **QNX** complies to `POSIX`, the typical process initiators like `exec`, `fork`, `spawn` and so on are present, although some have their limits towards compatibility (after all **QNX** is not **Linux**). For instance, the `fork` calls can only be done from processes containing only one thread.

The executable file format used is ELF. Also, it is possible to execute code sections from ROM/flash in place (XIP) so that all RAM is available for writable data.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

Below is a table with an overview of the scheduling characteristics.

	OS under evaluation
Supported memory models	Processes protected between each other, with multiple threads in one process.
Thread/process priority levels	256 levels (privileged priority levels included)
Max. number of processes	_SC_CHILD_MAX, can be set by the `sysconf()` call: the number of processes by userID. The hard limit is 4095.
Max. number of threads	The maximum number of threads in a process is limited to 32767.
Scheduling policies	Priority based scheduling for threads not belonging to the same priority level. Threads on the same priority: ***FIFO*** or ***Round-Robin*** (thread time-slice is always 4 OS clock ticks). ***Sporadic scheduling*** is an additional scheduling mechanism that overlays the standard thread priority scheduling model. A thread can have two priorities between which it switches dynamically; this is called sporadic scheduling.
Number of documented thread states	Ready, Running and Blocked. **QNX** uses different state names depending on the reason of the blocking. With these detailed block states, the total number of documented states is 21.
Number of undocumented thread states	N.A.

4.1.2 Memory Architecture

The memory architecture of **QNX Neutrino v6.5** is similar to the classical memory protection between processes just like in the traditional operating systems such as **Solaris**, **Linux**, and **Windows**.

Each process also has its own private virtual memory space; same addresses are reused in each process. Depending on the processor used, each user process can have between 2 to 3.5 GB virtual memory available.

Memory protection cannot be disabled as the memory architecture will depend on an MMU being available.

4.1.2.1 Boot ROM

When code is located on a boot ROM, it can be executed in place without using a RAM copy. Of course, this has its performance impact and as such it is used today only in rare occasions.

Today, booting will mostly occur by using NOR/NAND flash or block devices (hard disks, SSD).

Remark that even when you are running from a block device, if the same code segment is used in different processes, then the code needs only be present once in RAM.

4.1.2.2 Paging

QNX uses physical memory paging with 4-KB pages (similar to **Linux**). The RTOS manages the page tables from which the most frequently used pages are cached in the TLB of the CPU. Variable size pages are also supported on hardware where this is available. Variable size pages can improve performance by generating fewer TLB exceptions. A boot option can be used to turn off variable pages if desired.

In a context switch between processes, the RTOS manipulates the page tables so that the logical addresses, which may or may not be the same, point to the new physical addresses in memory. This operation is extra overhead when compared to inter-thread switching. **QNX** uses clever page table arrangements to minimize this overhead.

Although **QNX** does not support paging memory to disk, pages can still be unlocked in memory. For instance, **QNX** uses also an allocation on first write of physical memory (compared with **Linux**). The purpose of unlocked memory is to delay acquiring system memory and initializing it until an access to that area occurs. **QNX** supports `POSIX` memory locking so that a process can delay and potentially avoid the latency of fetching a page of memory. Once memory is acquired and initialized, it is set to locked state. Locked memory can be shared between processes. Super-locked memory is initialized memory that is private to a process. If you attempt to write to a locked memory page, then the kernel gets involved and may setup a private (super-locked) page if this is required. Memory locking can be controlled on start-up via ***procnto*** parameters or programmatically via system calls such as `mlock()`.

We use the ***procnto*** –mL option in all our tests so that all memory pages are acquired, initialized, privatized and super-locked right away. This prevents extra delays when unexpected and thus improves real-time behaviour.

On requests for larger physical continuous blocks of memory, **QNX** may invoke defragmentation of physical pages if needed. This process involves copying and remapping memory with the goal of coalescing free memory into larger blocks. Since other tasks may be running simultaneously, the defragmentation

algorithm takes into account that memory mappings can change while it is running. The algorithm takes also into account that certain pages such as those used by the kernel and hardware cannot be moved in physical memory. Memory defragmentation can be turned off via a ***procnto*** option if desired.

4.1.2.3 Heap

The heap is a standard heap system as used in any traditional operating system. A debug version of the heap library can be used to detect memory leaks and heap corruption issues such as under-runs and over-runs. The debug version of the heap library is not meant to be used in the final version of the product as it will influence real time behaviour.

	OS under evaluation
MMU support	Yes
Physical Page Size	4Kb, but large pages available (if supported by CPU)
Swapping/Demand Paging	Not available.
Virtual memory	YES (**QNX** requires a MMU to run)
Memory protection models	☺ Full virtual memory protection. Each process runs in its own virtual memory space

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.1.3 Interrupt Handling

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

The **QNX NEUTRINO RTOS v6.5** supports interrupt sharing. When an interrupt occurs, every interrupt handler attached to the hardware interrupt is executed in turn. The order of interrupts can be influenced by the order the interrupt handlers are attached (you can attach new handlers on the front or on the back of the list).

☺ 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.

Interrupt service routines can return NULL (= nothing to activate) or a pointer to an event to the microkernel. If the event is used, then the thread that attached the ISR can wait for the event to occur.

You can also avoid creating your own interrupt service routine and directly attach an event to an interrupt. In this case, the kernel automatically does a masking of the interrupt when delivering the event to the attached thread. After the interrupt-handling thread has dealt with the event, it must unmask the interrupt to re-enable it. This is a highly secure way of handling interrupts because there is no ***ISR*** called from the kernel.

☺ The interrupt handling model is made in order to be able to consider a device driver just like any other user process. It is therefore not linked in the kernel (that's why we call this a true microkernel architecture). Device drivers need only enough privileges to access device memory.

ISRs can also easily pass data through application-defined structures to the process that attached it (as the ***ISR*** actually runs in the process space that attached it). Attaching an event to an interrupt avoids writing an ***ISR*** and so there is no need to pass data from the ***ISR*** to the event servicing thread.

The use of ***semaphores*** in the ***ISR*** was intentionally blocked, because it is a less elegant solution for an RTOS with message based architecture.

	OS under evaluation
Handling	☺ Nested and prioritized
Context	The ***ISR*** runs in the context of the process that attached the interrupt handler.
Stack	The ***ISR*** has a special limited stack (normally around 200 bytes)
Interrupt-to-task communication	An event can be used from within the ***ISR*** to signal the Interrupt Servicing Thread (IST). This event can be used to pass data from the ***ISR*** to the servicing thread as well.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.1.4 System timer

The operating system clock timer (also called clock tick) is by default set on 1ms for processors running faster than 40MHz.

Nevertheless, it is possible to change this by a system call. Of course, it will be rounded depending on the hardware limitations for generating the clock interrupt.

All system timers have accuracy which is not better than the clock period, just like for any operating system.

The system timer can also be speed-up or slowed-down so it is possible to gradually synchronous time between devices (for instance by using SNTP protocol). This mechanism avoids abrupt time steps during run-time.

4.1.4.1 Hi resolution timers

QNX supports also hi-resolution timers (performance counters), depending on the hardware being used. These timers have an increase rate depending on the hardware. If such hardware is not present, the kernel will emulate one.

It is possible to retrieve the frequency of the hi-resolution timer (as it will depend on your hardware).

Remark that in a SMP system, these CPU timers depend on the CPU that the thread is currently running on.

4.1.4.2 Time-out timers

QNX Neutrino v6.5 has a special mechanism for timing-out on blocking system calls; the thread can request the kernel that it wants to time-out with a certain amount of time before performing the blocking call. This has some advantages, as the kernel itself will start the timer upon the system call. If the thread would start a timer before calling the blocking system call, then it could be pre-empted before actually issuing the system call (and thus the timeout could be caused by another high-priority thread).

☺ This mechanism is pretty unique in the RTOS world and prevents some developer headaches.

4.1.5 Synchronisation mechanisms

QNX is built around a ***Synchronous Inter-Process Communication*** messaging system. This is then also the most used and highly optimized mechanism to communicate between processes. This system behaves somehow like a library; function calls actually run in another process but the call does not return until the handling is finished.

Luckily, **QNX** supports also asynchronous messaging queues based on the `POSIX` standard. This is more useful in a loosely coupled designs where the messages are forwarded without acknowledgement (a fire and forget solution).

4.1.5.1 Protection mechanisms

QNX Neutrino v6.5 provides all protection mechanisms as foreseen by the different `POSIX` standards. Some of them are usable between threads only, some also between processes and even between remote processes (on a network).

These are the common protection mechanisms (see POSIX):

- ***Mutexes***: the same protection mechanism as defined in our evaluation terminology. It is used for protecting ***critical sections*** when manipulating shared data between threads. In **QNX**, ***mutex*** by default uses priority inheritance to avoid priority inversion. It is possible to disable priority inversion for a certain ***mutex***, but this should be a very bad idea. If you really believe that for a certain ***mutex*** no priority inheritance is needed, then you probably require a ***semaphore*** instead of a ***mutex***. ***Mutexes*** are normally used between threads only. They can be used between processes if the ***mutex*** is located in a memory region shared by the two processes. Indeed, if there is no shared data to protect (and thus a shared memory region between processes), then there are no ***critical sections*** and thus no need for a ***mutex***.

  It is important to remark that on most CPUs the kernel is not involved in the ***mutex*** handling until a ***mutex*** call changes the state of the caller or releases a thread. Only processors which do not have support for atomic instructions require kernel access for each ***mutex*** system call.

- ***Semaphores***: They do not use priority inheritance and can be easily used between processes. **QNX** also provides named ***semaphores*** to be used between network nodes. ***Semaphores*** should not be used to protect shared data (for which ***mutex*** are appropriate), but to synchronise threads.

Beside these common protection mechanisms, **QNX** also provides following POSIX mechanisms:

- ***Conditional variables***: wait within a ***critical section*** until something is true.
- ***Barriers***: a synchronization mechanism that lets you "corral" several cooperating threads, forcing them to wait at a specific point until all have finished before any one thread can continue.
- Sleep-on locks: a special kind of conditional variables.
- Reader/writer locks: these locks are used when the access pattern for a data structure consists of many threads reading the data, and (at most) one thread writing the data. These locks are more expensive than ***mutexes*** (as they are built upon conditional variables and ***mutexes***).

We refer to the POSIX standard and online **QNX** documentation for more information about these.

It is important to note that there are a lot of possibilities to choose from if it comes to synchronisation primitives. However, normally ***mutexes*** and ***semaphores*** will in most cases do the job.

In scope of the evaluation framework, the priority inheritance mechanism for ***mutexes*** is the most important feature.

4.1.5.2 Interlocked mechanisms

QNX Neutrino supports some atomic system calls for:

- adding a value
- subtracting a value
- clearing bits
- setting bits
- toggling (complementing) bits.

These work only locally (upon shared data).

4.1.5.3 Communication mechanisms

Again **QNX** provides most available Inter-Process Communication mechanisms foreseen in the different POSIX standards.

4.1.5.3.1 Synchronous message passing

The most important mechanism is the message passing interface as it is used by the different system calls. This message passing mechanism uses synchronous message-based priority server handling of messages, making such calls transparent if they were handled in another process or within the same thread (same priority is used).

This message passing system uses channels to send messages to, so that messages are not directly sent to threads but to channels.

4.1.5.3.2 Pulses

Pulses are an asynchronous system to pass messages between different processes. However, they can only be used for small payloads (8-bit pulse code with a 32-bit data argument).

4.1.5.3.3 Events

Pulses, POSIX signals and even interrupts are all based on the **QNX** internal asynchronous event handling system.

4.1.5.3.4 POSIX signals

For this we refer to **POSIX**, most developers will know this mechanism from building **UNIX** applications.

Furthermore the signals are defined in three regions:

- Classical POSIX signals (including traditional **UNIX** signals)
- POSIX real-time signals
- Some special-purpose **QNX** signals.

4.1.5.3.5 Asynchronous message passing

Besides the synchronous message passing, **QNX** also supports the POSIX asynchronous message queues. This is done by an optional resource manager the ***mqueue***.

☝ There's a fundamental difference between the **QNX** synchronous messages and POSIX message queues. Synchronous messages block copy their data directly between the address spaces of the processes sending the messages while POSIX messages queues, on the other hand, implement a store-and-forward design in which the sender does not need blocking and may have many outstanding messages queued. The synchronous messages also inherit the priority from the message sender, which is not the case for the POSIX message queue.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

Asynchronous message passing following `POSIX` use a file like system: a path to open the queue and read/write to receive and send data on it.

4.1.6 Specialities

Here we list some special features that can be used with **QNX**.

4.1.6.1 Adaptive partitioning

Adaptive partitioning is an additional scheduling mechanism that overlays the standard thread priority model. It allows creation of separate partitions to which processes or threads are assigned, and guarantees that those partitions receive the configured amount of CPU if they need it. Unlike a hard partitioning system, idle time is not wasted, but is redistributed by the scheduler to the partitions that need it.

This can be used for instance to tackle denial of service attacks.

The adaptive partitioning mechanism uses a sophisticated micro-billing scheme to track usage against each partition. It includes features like:

- Dynamic creation of partitions and partition lock-down for configuration security
- Sliding time windows for partition calculation with a user-configurable window size
- Assignment of individual threads to partitions outside of a processes main partition
- Critical threads are allowed to borrow against a partition's limit to ensure they always execute
- Appropriate client/server billing where server's time is billed to a client's request: this is similar like priority passing when requesting a work item from a server.
- Ability to change application execution partitions at run time without changing source code.

There cannot be put constraints on memory resources.

Adaptive partitioning can be used to assist in integration, by making sure all software contributors are operating within their given budgets. It can also guarantee that low-level processes are not completely starved with unfound bugs, unusual circumstances or attack instances. Things like command-line or debugger access, logging facilities, or administrative controls can be allocated into a small partition to be certain that they will always run, even when the system is overloaded with high-priority tasks.

Remark however that the designers have to know what they are doing, as wrong partition configuration can cause deadline misses (by not allowing critical threads to run the time it needs). Such things can be avoided by a correct set-up.

4.1.6.2 High Availability (HA)

There are also some high availability features. Already, due to the message system and micro kernel with device drivers and other servers having their protected memory, **QNX** has a robust design.

The HA feature that can be added is to transparently recover a server (and its listen channel), without any client noticing this.

In such cases a server failure or reboot can be detected by a HA manager, which restarts the server. The restarted server can then handle the pending request.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

☺ The nice thing about this mechanism is that the client does not need to know that the service has been restarted nor it has to check if some failures occur as this is done automatically.

4.1.6.3 PPS (Persistent Publish/Subscribe)

This is an extensible service that offers persistence across reboots and is a key mechanism to enable developers to build loosely connected systems using asynchronous publications and notifications.

4.1.6.4 Transparent Distributed Processing

Another feature is Transparent Distributed Processing using ***Qnet*** which extends inter-process communications transparently over a network of micro-kernels. As such, distributed systems are easier to build. Further, ***Qnet*** can be used to add redundancy by multi-path links.

4.1.6.5 BMP (Bound Multi-core processing)

BMP is an extension on SMP that allows the developer to lock individual threads or processes to a specific CPU. This capability is useful to support legacy applications that use poor techniques for synchronizing shared data (typically the use of implicit locking by priority, which is of course of no use on SMP systems).

This is very similar to setting CPU affinity. However, **QNX** provides the ability to start an application by command line on a specific core (at runtime, thus without adapting the source code).Additionally, with BMP, new threads created by bound threads will inherit the affinity mask of their creator.

4.1.6.6 Instant Device Activation

Instant Devices Activation allows a developer to set up a “mini driver” to respond quickly to external stimulus when the system boots. A typical scenario involves a device that sends a message within 30 milliseconds after power is turned on. Instant device activation can save on hardware costs by reducing the need for extra chips and circuitry to handle these scenarios.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.2 API richness

*The **QNX NEUTRINO RTOS v6.5** 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. There is only a lack for fixed block size memory partitions.*

IMPORTANT NOTICE: the purpose of the tables below is only to make an inventory of the kernel related APIs. They make an inventory of basic real-time objects and features present in the `POSIX` and OS proprietary interfaces.

While interpreting these results, the reader should keep in mind that these tables cover a strictly defined set of system calls only. As it is very hard to compare the different API for the different operating systems, no points are given for this section. The reader should use this section to check if the API calls he needs for his application are available or not.

In general, the more system calls there are available, the easier it is for the programmer to find the correct call for his application, without writing a proprietary library with for instance wrapper calls…

4.2.1 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	✓

Thread management	YES
Lock thread in memory	✓[1]
Disable scheduling	✓

4.2.2 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	✓

Interval timer	YES
Timer expires on an absolute date	✓
Timer expires on a relative date	✓
Timer expires cyclique	✓
Get remaining time	✓
Get number of overruns	✓
Connect user routine	✓

4.2.3 Memory Management

Fixed block size partition	NO[2]
Set partition size	-
Get partition size	-
Set memory block size	-

[1] Locking in memory is done by mlockall calls on a process basis or on bootup by procnto options.
[2] The heap can be configured with certain fixed block sized bands.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

Fixed block size partition	NO[2]
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	-

Non-fixed block size pool	YES[3]
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	-
Lock/unlock pool in memory	✓
Lock/unlock block in memory	✓

[3] Standard heap (malloc).

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.2.4 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	✓

4.2.5 Synchronization and Exclusion Objects

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

Binary semaphore	NO
Set initial value	-
Share between processes	-
Wait - blocking	-
Wait - non blocking	-

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

Binary semaphore	NO
Wait - with timeout	-
Post	-
Get status	-

Mutex	YES
Set initial value	✓
Share between processes	✓[4]
Priority inversion avoidance mechanism	✓
Recursive getting	✓
Thread deletion safety	✓[5]
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Release	✓
Get status	-
Get owner's thread ID	-
Get blocked thread ID	-

Conditional variable	YES
Pend non blocking	✓
Pend with timeout	✓
Pend in fifo / priority order	✓
Broadcast	✓

Event flags	NO
Set one at a time	-
Set multiple	-
Pend on one	-

[4] If in shared memory
[5] Can be done by a non-posix mutex wakeup.

Event flags	NO
Pend on multiple	-
Pend with OR conditions	-
Pend with AND conditions	-
Pend with AND and OR conditions	-
Pend with timeout	-

POSIX signals	YES[6]
Install signal handler	✓
Detach signal handler	✓
Mask/unmask signals	✓
Identify sender	✓
Set destination ID	✓
Set signal ID	✓
Get signal ID	✓
Signal thread	✓
Queued signals	✓

4.2.6 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	✓

[6] On top of these signals, QNX supports all of the enhanced POSIX real-time signals.

Queue	YES
Receive – with timeout	✓
Send - with ACK	✓
Send - with priority	✓[7]
Send – OOB (out of band)	✓[8]
Send - with timeout	✓
Send – broadcast	-
Timestamp	-
Notify	-

Mailbox	NO[9]
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	-

[7] Queues are always prioritized and FIFO when messages have the same priority.
[8] Indirectly by using priorities
[9] A mailbox is in fact nothing more than a message queue that can store no more than one message. It is included for the sake of completeness, but most operating systems do not explicitly support it anymore.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.3 Documentation

OS Documentation 0 9 10

It has been always one of the best documented RTOS on the market and still it is.

Just like before, the documentation does a serious job by providing a general overview of the system and its architecture.

The first task a user should do is to use the Quick start Guide: “10 Steps to Developing a **QNX** Program”, which guides him how to setup a **QNX** PC host/target and to run/debug a simple “hello” application. It is a very simple document adapting the concept of “do this, click there” which helps the user to get started-up quickly in developing his applications. Remark that you can run **QNX** as well in a **VMWare virtual machine** which is a very useful step to get things starting up quickly on your machine. It makes the development procedure easy as well to run your code already on a target, even without any specific hardware available yet.

For more detailed information about the operating system, we suggest to start by going through the system architecture section of the documentation.

Everything in the documentation is well written and all special things are highlighted (for instance, that a certain feature is available since version X). Further, if needed, extra insight is given concerning real-time behaviour.

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

The support is handled via a support website where it is easy to track issues and handle them. For instance, it is possible to attach files and similar issues.

We used the support site a couple of times and it worked well.

There is Foundry27 as well, which is a collaboration instrument for developers working with **QNX** and open source related. For instance you will find code and examples how to run **QNX** as a QEMU host or guest. The number of projects are however limited.

4.4 OS Configuration

OS Configuration 0 [8] 10

*Installation is quick and simple if a BSP is supplied from **QNX**. Creating and configuring a custom **QNX** image is done in the IDE or by text-based build files. However building a new BSP isn't easy. Anyhow, all BSPs and drivers are delivered in source code and can be ported to your platform. Only the kernel and libraries aren't delivered in source code*

4.4.1 OS Boot options

Booting up a target which runs **QNX** is a process of three steps:

- The ***Initial Program Loader*** (IPL), which:
  - set up the crucial hardware (for instance DRAM timings)in order for any software to run
  - set up a device used to load the OS (and application)
  - loads the OS in memory
  - Jump to the first address of the loaded OS (which is the start-up program).
- The start-up program, which
  - sets the hardware configurations needed by the kernel in predefined structures
  - Searches for the different bootable configurations (multiple may be installed!)
  - mount the boot file system (mapped into memory, if downloaded OS) and start the kernel
- The kernel then runs the start-up script which is used to load supported devices and start the applications

For example, ***IPLs*** are delivered in source code (e.g. for each supported board), which may be used to build yours. However, setting up the hardware isn't easy and has to be documented by the board manufacture.

The ***IPL*** does not need to be the one from **QNX**! Any piece of code that sets up the hardware and downloads and can start a binary will be sufficient. For instance, the BIOS on a PC can be used, but also u-boot or other ROM monitors can be used.

The learning curve for understanding the start-up mechanism is steep, which is the case with all other RTOSs.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.4.2 OS Configuration

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

The **System Builder** tool works reasonably 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 binaries and shared objects (DLLs): this is excellent! The **System Builder** will also automatically add the shared libraries needed by the added module in your target system configuration.

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.

4.4.3 Building your own BSP

☺ When installing **QNX 6.5 Neutrino**, you will notice that a lot of BSP code is delivered in source code (or can be downloaded from Foundry27).

Relating to licensing, the licensing information associated with any specific driver is specified in the headers of the source file. **QNX** typically releases driver source for which they own the IP under Version 2.0 of **Apache** which allows for broad rights.

If required, you can apply for source access to other source code as well. There are cases where 3rd party source is used as part of **QNX** solution where they don't have rights to distribute the source. Example areas include but are not limited to wireless drivers, accelerated 2D/3D graphics and codecs.

The major thing that is not delivered in source code is the process manager (***procnto***): the microkernel of **QNX**.

If a driver is supported for one target by **QNX**, then it can be easily used on any other target (just recompile the code with the correct compiler).

Of course, you can also make your own device driver, but this is another issue that is not discussed in this document.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

4.5 Internet components

Internet components	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 Web Browser based on the open source WebKit:
  - full HTML 4.01 support,
  - CSS 2.1 and parts of CSS 3 support,
  - javascript (1.5, 1.7 and ECMA-262 3rd edition),
  - Document Object Model (DOM Level 1 and 2)
  - XML parsing (XHTML 1.1 and XML)
  - XSLT and XPath 1.0
  - HTML canvas and AJAX
  - SMIL 2.2 or 2.3
  - Image formats: jpeg, gif, png
  - HTTP 1.1, HTTP Cookies
  - SSL3/TLS1.0
  - SSL Root Certificates Support: VeriSign, Entrust, Thawte, Baltimore.
  - Support of externally supplied fonts
  - Different character sets
- Another web browser typically used as a help viewer.
- 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.
- As it is `POSIX` compliant, it is possible (and not too difficult) to build open source applications for **QNX**. For instance, in our evaluations we used `ffmpeg` for some tests. We had no problem to build this for **QNX**.
- The apache web server is also available for **QNX**.
- Further **QNX** has its own SQL database, which is based on `sqlite`.

4.6 Development tools

Development tools	0								9		10

QNX Neutrino 6.5 *provides both command line and IDE based development tools on a variety of host platforms. We used and tested the* **QNX Momentics IDE** *based on the (extensible) Eclipse Framework, on both a* **Windows** *and on a* **Linux** *host.*

Since our previous evaluation (6.3), the **Eclipse/Momentics** *combination has become more resource friendly. It consumes around the half of the RAM it consumed in our previous evaluation. Also general performance and usability is improved.*

The toolset has become more intuitive, and is one of the better development tools around for cross development and debugging. Well done!

4.6.1 Basic Development

We used the **Eclipse Framework** to build our test system. Tools for both self-hosted (**QNX** host, with Photon GUI) and cross development are available both on **Windows** and **Linux** variants. These toolkits contain the most commonly used tools. The tools are based on the GNU toolchains (also the debugger).

The tools work as well on a **Linux** distribution (we tested it on **Ubuntu** 9.10, as it was officially supported by **QNX**) and **Windows**. Both have the same look and feel as the **java** based **Eclipse IDE** is used as base.

We also created a Virtual Machine (VM) running the **QNX** RTOS and used it as remote debug target. **QNX** runs without any problem in the VM (using VMWare workstation). Remote debugging is very intuitive and works correctly, although it can sometimes break the connection if you do quickly launch (in debug mode) and stop applications without leaving enough time to handle this remotely.

As device drivers are application processes as well, they can be debugged similar to any other application. We found this very useful for debugging our PCI driver: by debugging the PCI server, we found the correct way to init the BAR regions of our device.

As the BSP is given in source code (and thus the PCI server as well), we could easily step through the code.

We have one complain when debugging BSP: it seems very difficult to make debug builds without optimizations for these BSP drivers. For other applications this was easy, but not for the delivered BSP drivers.

☺ In our previous report we indicated the heavy use of memory resources by **Momentics** and **Eclipse**. This has been addressed: we had no problem to use the **IDE** on a (virtual) **Windows** machine with only one core and 512MB ram (and the same for a **Linux** host).

Although one of our previous reports comparing Microsoft's **Visual Studio** and (standard) **Eclipse** (can be found on downloads sections of dedicated systems website) was not very positive about **Eclipse** when it

comes to **C/C++** development, the way **QNX** configures **Eclipse** and adds plug-ins makes this IDE much more useful.

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

- Photon Developer's toolkit to develop Photon (a small widget toolkit) 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**	✓	✓	✓	✓
Java	**Yes**	✓	✓	✓	✓
Ada	**No**				
Assembler	**Yes**	✓	✓	✓	✓
Linker					
Incremental	**Yes**	✓	✓	✓	✓
Symbol table generation	**Yes**	✓	✓	✓	✓
Development tools					
Profiler	**Yes**	✓	✓	✓	✓
Project management	**Yes**	✓			✓
Source code control[10]	**Yes**	✓	✓	✓	✓

[10] The Momentics IDE supports SVN (default), CVS and other 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.

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Revision control	**Yes**	✓	✓	✓	✓
Debugger					
Symbolic debugger	**Yes**	✓	✓	✓	✓
Thread sensitive debugging	**Yes**	✓	✓	✓	✓
Mixed source and disassembly	**Yes**	✓			✓
Variable inspect	**Yes**	✓	✓	✓	✓
Structure inspect	**Yes**	✓	✓	✓	✓
Memory inspect[11]	**Yes**	✓	✓	✓	✓
Target connection					
JTAG	**Yes**	✓			✓
BDM	**No**				
Serial (via ROM-Monitor or app?)	**No**				✓
Network (via ROM-Monitor or app?)	**Yes**	✓			✓
System analysis tool					
Tracing	**Yes**	✓	✓	✓	✓
Thread information	**Yes**	✓	✓	✓	✓
Interrupt information	**Yes**	✓	✓	✓	✓
Loader					
TFTP boot loader	**Yes**	✓	✓	✓	✓
Serial boot loader	**Yes**	✓	✓	✓	✓

[11] 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.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Load separate modules	**Yes**[12]	✓			✓

4.6.2 Extra Tools provided

Besides the normal build/debug environment, **QNX** did a good job to integrate as well different tools inside Eclipse. These tools can be very helpful for detecting memory faults, performance problems, system behaviour and much more. In this section we go a bit deeper into this toolset.

Remark that for all tools, **QNX** provides example projects combined with a cheat sheet on how to configure and use a certain tool. As such you can quickly learn how to use these tools. A good tutorial is not everything; the toolset has to be intuitive enough so you can apply it easily with your project as well. Also this is something **QNX** took into account. You get quickly experienced how to use these tools.

If you are limited to the freely provided tools, **Eclipse** is not a good choice. However, the way **QNX** integrates its plug-ins in this IDE shows that such open framework has serious potential and can easily compete with proprietary tools.

4.6.2.1 Memory Analysis tools

The first tool we are going to discuss is the Memory Analysis tool. This tool can be used to trace memory events like allocations and releases, which is useful for detecting memory leaks.

The memory leaks have also their back stack traces logged at the moment they were allocated. You can compare this with the Valgrind memory analysis, but with a graphical user interface and integrated in the tool-chain.

Detecting memory leaks is of course very important in embedded systems which do not have only a smaller memory footprint compared with desktop machines, but they should run all the time without any reboots!

Another aspect of the tool is that it detects usage errors. For this it uses as well some instrumentation of different string and memory manipulation routines which can easily cause errors when wrongly used. Further, boundary checks and heap integrity checks can be done. Not all of these kinds of errors can be redirected to the line where it is introduced (like buffer overruns); however you will find the place in the source where the block was allocated.

These tools also gather allocation information, so you can finely tune the heap parameters for your application.

4.6.2.2 Mudflap

This is another memory analysis tool (an open source extension on **gcc**) which will instrument the code. It cannot be combined with the other memory analysis tool.

[12] You have direct access to the remote filesystem by the IDE via the network (as long qconn runs on the target).

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

QNX did also a good job here by integrating the output (which is text base) to a graphical output window and to link it back to the source code.

4.6.2.3 Code coverage

Code coverage uses the **gcc gcov** coverage output. Enabling the code coverage on the build requires manual intervention by changing the compiler settings in the makefile.

On the other side, adding the tool in your run environment will retrieve the coverage logs integrated in the IDE. As such, the code coverage results are well integrated. You can directly verify in your code which code lines are not covered yet.

Also it is very easy to combine multiple code coverage sessions in one overall result. So you can detect which code lines are not yet executed across all your tests.

Last, it is possible to generate an HTML report about the results.

4.6.2.4 Application profiler

There is as well an Application Profiler, which can be used to sample time usage (by statistically sampling gathering from the kernel) of the different components in the application. It can as well instrument the code to retrieve call graphs and number of calls.

Both systems can be combined.

It is also possible to compare two running versions (for instance after you made some code optimizations).

4.6.2.5 System profiler

On a higher level, it is possible to profile the system by using the instrumented kernel (this has of course some performance impact).

Of course, profiling like this will generate a huge number of events. The problem is to find the problem case in the pile of information. Luckily there are good ways to filter out events (limiting what is displayed) and complex event search tools available.

Following information can be gathered from this tool (in short any kernel event):

- CPU usage
- CPU migration
- IPC
- All types of events (interrupts, messages, locks, scheduling)
- All kinds of statistics

Events can be viewed in a table, but as well in a timeline. Further, there are different ways to show the information.

In QNX neutrino 6.5, there is now support to generate custom events. This can be very handy to link traces with application events so you can quickly find the part of the traces you are interested in. As the kernel does not has any knowledge on the source code involved, it is not possible to find for instance which ***mutex*** call (line in the source code) generated the event.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

The fact that such traces can be very useful was proven during our tests. We had a problem with an interrupt test while loading the processor. By using these kernel traces, it was detected that the interrupt was shared with the USB driver. The USB driver only enabled the interrupt again in its deferred handler in a thread. As we were loading the system, this USB thread (having lower priority than ours) could not run and as such the interrupt stayed masked. Such complex issues are hard to find if you do not have such kernel event tracing available.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

5 Appendix A: Vendor comments

All vendor comments were integrated in the document as there were no disagreements.

http://www.dedicated-systems.com
email: m.timmerman@dedicated-systems.com

6 Appendix B: Acronyms

Acronym	Explanation
API	Application Programmers Interface: calls used to call code from a library or system.
BSP	Board Support Package: all code and device drivers to get the OS running on a certain board
DSP	Digital Signal Processor
FIFO	First In First Out: a queuing rule
GPOS	General Purpose Operating System
GUI	Graphical User Interface
IPC	Inter-Process Communication
IDE	Integrated Development Environment (GUI tool used to develop and debug applications)
IRQ	Interrupt Request
ISR	Interrupt Servicing Routine
MMU	Memory Management Unit
OS	Operating System
PCI	Peripheral Component Interconnect: bus to connect devices, used in all PCs!
PIC	Programmable Interrupt Controller
PMC	PCI Mezzanine Card
PrPMC	Processor PMC: a PMC with the processor
RTOS	Real-Time Operating System
SDK	Software Development Kit
SoC	System on a Chip