

RTX 4.2 Evaluation Executive Summary

The following article is the long-awaited executive summary report of our evaluation of RTX 4.2, the real-time extension to Windows NT from VenturCom Inc. In it we give an overview of the system architecture, API richness, performance, and support available.

INTRODUCTION

During the latter part of 1998, Real-Time Consult officially started an independent evaluation program for RTOS. Windows NT workstation was studied first and quickly followed by the leading real-time extensions for NT, namely:

- RTX 4.2 from VenturCom, Inc.
- INtime 1.20 from Radisys Corporation Ltd.
- Hyperkernel 4.3 from Imagination Systems, Inc.

The three complete evaluation reports, as well as a full comparison report highlighting the decision critical information for all three products are available for sale on our website (<http://www.realtime-info.com>).

This article presents an executive summary of the results from our evaluation of RTX 4.2, the windows NT real-time extension from VenturCom, Inc.

ARCHITECTURE

RTX 4.2 adds a real-time subsystem (RTSS) to

Windows NT. RTSS uses the NT device driver concept for its implementation. This is shown in Figure 1.

By implementing the real-time subsystem as an NT device driver, VenturCom opted for a highly integrated design. This approach facilitates communication between RTSS and Win32 processes.

The same approach also has drawbacks, especially concerning reliability of the subsystem. If NT crashes for example, the integrity of the entire NT environment could be compromised. Since the real-time subsystem runs in the same memory space as NT, it could become unstable too and keep it from shutting down properly.

All the RTX application "processes" are linked and loaded as device drivers and therefore run in kernel mode.

Tasks

RTSS is a multi-process, multi-thread subsystem. Every process has at least one thread (the primary thread). An RTSS thread runs at one of 128 distinct priority levels.

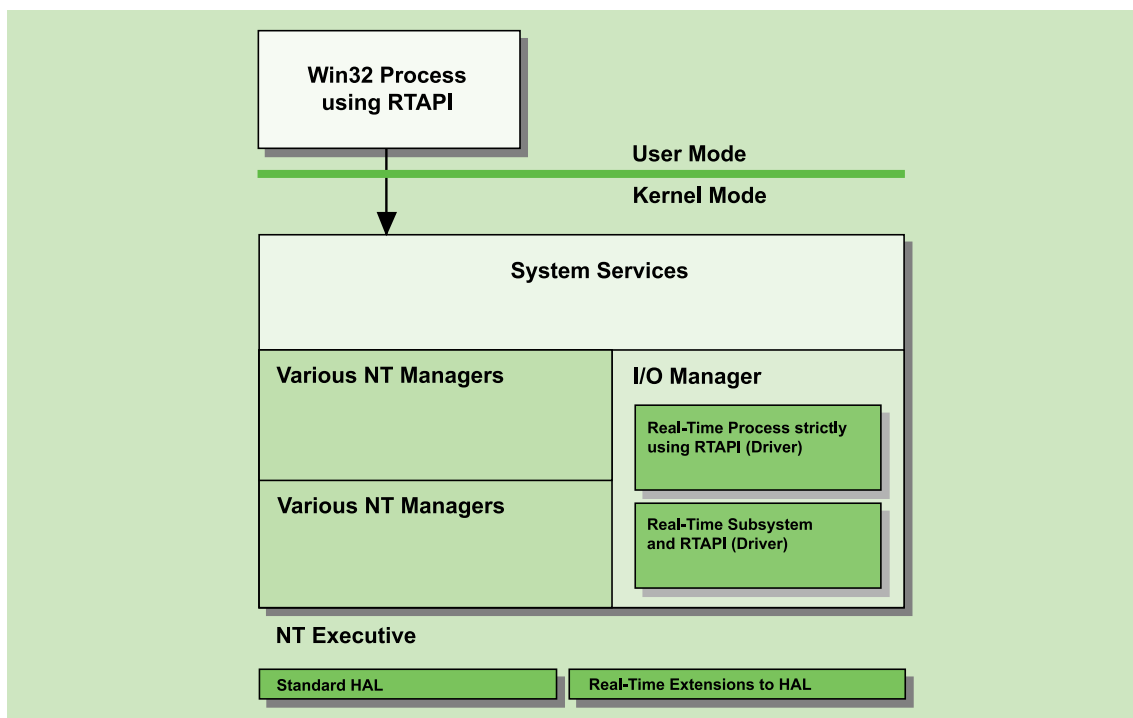


Figure 1: Integration of RTX 4.2 into Windows NT 4.0

RTSS processes are not POSIX compliant processes, since they all share one common address space. However, an RTSS process can still contain several threads, which can share object handles. This is not the case for threads in different processes. That is why Real-Time Consult has introduced the name "mini process" to designate RTX processes.

The RTSS environment has no distinct priority classes, so all the RTSS threads compete for CPU time using the thread priority only (priority based pre-emptive scheduling). Threads of equal priority are scheduled in FIFO order. The threads are not subject to time-sliced sharing of the processor.

There are some severe limitations on the number of "mini processes" that can exist concurrently in the RTSS subsystem, and how they can be created. This application segment limitation brings into doubt the capabilities of RTX 4.2 in particular types of applications. For a more detailed discussion, the reader is referred to the full evaluation report.

RTX 4.2	
Model	Threads & "Mini Processes"
Priority levels	128
Max. nr of tasks	Limited by available memory
Scheduling policies	Prioritized FIFO: the running thread executes until it: <ul style="list-style-type: none"> • is pre-empted by a higher-priority thread; • voluntarily gives up the processor
Number of documented states	?

Table 1: RTX 4.2 Task handling properties

Memory

The RTSS subsystem and all RTSS application "mini processes" are loaded as NT device drivers. Device drivers by default reside in the non-paged memory pool (locked memory), so they cannot be swapped out to

RTX 4.2	
MMU	Required
Physical page size	4KB
Paging/Swapping	No swapping
Virtual Memory	Not supported
Memory protection models	No memory protection. All user and system threads share the same memory space.

Table 2: RTX 4.2 Memory Management properties

disk. The heaps of all RTSS "mini processes" are allocated from the non-paged memory pool. RTX 4.2 memory allocation routines always allocate locked memory.

There is no protection between the memory spaces of the different RTSS "mini processes" and NT system processes, since device drivers in NT share one common address space.

RTX is not a standalone operating system, and therefore does not have its own services to manage memory, but instead relies on Windows NT for that. For a demonstration on how this makes the system's memory management unpredictable and why this is unacceptable in certain types of applications, the reader is referred to the full evaluation report.

Interrupts

RTX 4.2 uses a thread interrupt model. When the application attaches to a certain interrupt vector, the system creates an ISR and an IST (Interrupt Service Thread). As soon as the interrupt occurs, the ISR signals an event on which the IST is waiting. When the IST is awoken by its event becoming signaled, it calls the user-provided interrupt handler.

RTSS interrupts have priority over NT interrupts. When an interrupt occurs, the current interrupt source, as well as interrupts from lower priority sources are masked until the IST finishes executing.

RTX 4.2	
Handling	Nested, prioritized
Context	Interrupt handler runs in its own context
Stack	Interrupt handler has its own stack
Interrupt-to-task communication	Yes
Min. RAM	?

Table 3: RTX 4.2 Interrupt handling properties

API RICHNESS

To assess the API richness, we created a list of features for the most common system calls and compared it with the available system calls in RTX 4.2. Table 4 gives an overview of all the categories and the score (in percentage points) that was obtained. For a breakdown of the categories into individual features and system calls, the reader is referred to the full evaluation report.

This table should not be misunderstood. The RTX API has system calls that are not in our list, and are therefore not represented in Table 4.

An average percentage of 30% was obtained. The average percentage does not include any weight factors, it is simply the average of each category's score.

RTOS EVALUATIONS

MECHANISM	RICHNESS
Thread Management	35 %
Clock	14 %
Interval Timer	83 %
Fixed block size memory partition	0 %
Non-fixed block size memory pool	54 %
Interrupt Handling	38 %
Counting Semaphore	80 %
Binary Semaphore	0 %
Mutex	75 %
Conditional Variable	0 %
Event Flags	38 %
POSIX Signals	0 %
Message Queue	0 %
Mailbox	0 %
AVERAGE PERCENTAGE	30 %

Table 4: API Richness

INtime 1.20 from Radisys Corporation for example, received a total average percentage of 31 %.

As can be seen from Table 4, the API is missing some basic features like mailboxes and message queues.

PERFORMANCE TESTS

Interrupt latencies

For this test, we measured two latencies:

- Interrupt Latency (task to interrupt handler): The time elapsed between the execution of the last instruction of the interrupted thread and the first instruction in the interrupt handler.
- Interrupt Dispatch Latency (interrupt handler to task): The time needed to go from the last instruction in

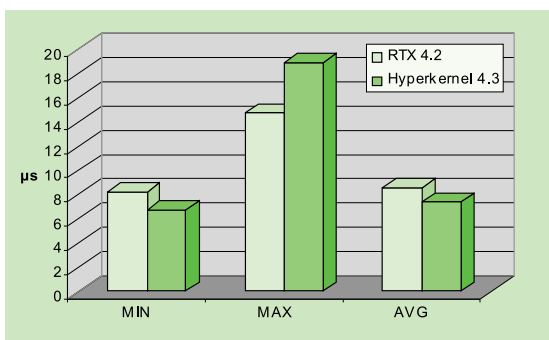


Figure 2: Interrupt Latency - RTX 4.2 and Hyperkernel 4.3

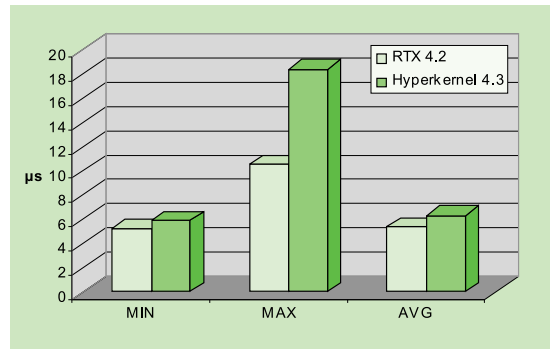


Figure 3: Interrupt Dispatch Latency - RTX 4.2 and Hyperkernel 4.3

the interrupt handler to the next task scheduled to run.

Figure 2 and Figure 3 display the minimum, maximum and average values for both interrupt latency and interrupt dispatch latency. The test results for Hyperkernel 4.3 from Imagination Systems Inc are included for the sake of comparison.

Interrupt handling in RTX 4.2 is pretty swift. This is partially due to the fact that in RTX 4.2, all application and system threads share one common address space, which results in faster context switches and interrupt handling. The drawback of this approach is in reliability and security.

We would like to point out to the reader that the above tests were designed in such way that an NT thread was executing while the interrupt occurred, thereby provoking a switch from NT to the RTX subsystem to let the interrupt handler execute. Due to the high integration of the RTSS subsystem in Windows NT, this switch is a fast operation.

Priority inheritance

RTX 4.2 has a priority inheritance mechanism, which is absolutely crucial for an RTOS to have. We tested this by creating a situation with 3 threads where the priority inversion problem occurs: a high priority thread wants to acquire a mutex that is owned by a low priority thread. A medium priority thread keeps the low priority thread from running and releasing the mutex so that the high priority thread can't acquire it.

For a detailed description and flow chart of the test, the reader is referred to the document "report definition and test plan", which can be downloaded for free from our website (www.realtime-info.be/eval).

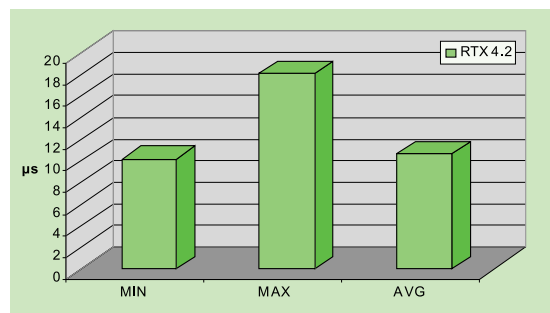


Figure 4: Priority Inheritance

AD VMIC

RTOS EVALUATIONS

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 results for the test are displayed in Figure 4. The priority inheritance mechanism seems stable.

TOOLS & DEVELOPMENT METHOD

The RTX application can first be prototyped as a Win32 application. The benefit of this is that the standard debugging tools available in Microsoft Visual Studio can be used to debug the prototype. A lot of developers are used to this environment and can be very productive in finding non real-time bugs.

During the second step, the Win32 application can be ported to RTX. This is a simple operation, since both Win32 and RTX support the full RTX API (albeit with different response times and performance characteristics). When debugging the RTX application however, kernel-level debuggers (e.g. Microsoft WinDbg) have to be used. While these debuggers still have nice features, debugging kernel level applications is substantially different from debugging conventional applications. Developers not accustomed to kernel level debugging will need time to get worked in.

DOCUMENTATION & SUPPORT

The documentation is clear and easy to read. The manuals are also available on the VenturCom web site (www.vci.com) in PDF format, and can be downloaded free of charge.

Readers who are interested in knowing all the architectural and design intricacies of RTX 4.2 may find the documentation a bit restrictive, but issues that pertain to general usage are covered thoroughly. The descriptions of the API system calls are precise and detailed, which is very important to application developers.

VenturCom's technical support is good. While executing tests on RTX 4.2, we contacted the tech support team a few times via email and got a response within 24 hours.

CONCLUSION

The RTSS subsystem is implemented using the Windows NT device driver concept. Due to this high level of integration, there is a lot of synergy between Windows NT and the real-time subsystem, facilitating communication and development of real-time applications. The drawback is in security and reliability: all RTX system and application threads share one common address space with other Windows NT device drivers. A faulty RTX application or NT device driver can easily compromise the integrity of both systems.

During our tests, we found RTX 4.2 to exhibit predictable real-time behavior in most cases. However, because of the memory management issue and an application segment limitation, RTX 4.2 is not appropriate for use in certain types of real-time systems. For a more detailed discussion on these issues and the impact they might have on a particular application, the

reader is referred to the full evaluation report.

OTHER PUBLICATIONS AND SERVICES

As mentioned at the outset, the complete evaluation reports for Windows NT 4.0, RTX 4.2, Hyperkernel 4.3 and INtime 1.20 are commercially available on our website.

A comparison report is also available for readers who are interested in how RTX 4.2 is situated against other Windows NT real-time extension products.

Real-Time Consult is currently evaluating VxWorks 5.3.1, QNX 4.25 and pSOS 2.2.6. Evaluation reports for these products will be available near the end of March 1999.

The evaluation reports are intended for everyone who is in one way or another involved with dedicated systems technology. This obviously includes the system design engineers and application developers, who need to have a detailed understanding of how the product behaves in a real-time environment, but the audience also includes managers and project leaders who need to make strategic decisions like which RTOS to use, and how it will affect the overall execution of the project.

Finally, Real-Time Consult also perform feasibility studies and product validations to meet the specific demands of individual customers. For additional information please contact our offices directly. ■

Dr. Martin Timmerman has a degree in Telecommunications Engineering from the Royal Military Academy (RMA) Brussels and received a Doctorate in Applied Science from the Gent State University (1982) in Belgium. In 1983 he transferred to Computer Engineering and set up the System Development Centre (SDC) at RMA. He gives general courses on Computer Platforms and more specific courses on System Development Methodologies. He is a consultant to the Joint Staff of the Belgian Armed Forces in areas concerning Information System Methodologies and CASE tools and he is the Belgian representative in some NATO technical commissions. Outside the RMA, Martin is known for his audits, reviews and seminars, and for his two companies Real-Time Consult and R.T.U.S.I., where he makes use of his considerable knowledge of the Real-Time world. Real-Time Consult is the publishing house responsible for Real-Time Magazine, an International magazine about Real-Time system development. Real-Time User's Support International (R.T.U.S.I.) provides hardware and software support services and is involved in project engineering for real-time systems.

Bart Van Beneden has been with Real-Time Consult since 1998 where he is involved in the RTOS evaluation program of Real-Time Magazine as a project manager. He received his degree in computer science at the Free University of Brussels. Before joining Real-Time Consult, he designed multi-media applications with LaserMedia Inc.