

INtime 1.20 Evaluation Executive Summary

The following article is the executive summary report of the evaluation of INtime 1.20, the real-time extension to Windows NT from Radisys Corporation Ltd. It gives an overview of the system architecture, API richness, performance and available support.

INTRODUCTION

During the summer of 1998, Real-Time Consult officially started an RTOS evaluation program. First, Windows NT and the real-time extensions to Windows NT were studied. The evaluation reports for the following products are currently available:

- RTX 4.2 from VenturCom, Inc.
- INtime 1.20 from Radisys Corporation Ltd.
- Hyperkernel 4.3 from Imagination Systems, Inc.
- VxWorks/x86 5.3.1 from WindRiver Systems Inc.
- pSOSystem/x86 2.2.6 from Integrated Systems Inc.
- QNX 4.25 from QNX Software Systems Ltd.

The evaluation reports, as well as comparison reports highlighting the decision critical information are available on our website (<http://shop.realinter.net/rtshop/>).

This article presents an executive summary of the evaluation report of INtime 1.20 from Radisys Corporation Ltd.

ARCHITECTURE

INtime 1.20 uses the hardware task management facilities of the Intel architecture available when the processor runs in protected mode. This mechanism allows the processor to dispatch, execute and suspend hardware tasks. The difference between software and hardware tasks is that software tasks are scheduled and dispatched by the operating system, whereas hardware tasks are scheduled by the operating system and dispatched by the processor.

Standard Windows NT runs within a single hardware task, using a software dispatching mechanism for task management. When INtime 1.20 is installed, the extension will create a separate hardware task during boot time that contains the entire INtime system (kernel and user processes).

The benefit of this approach is in security and stability. NT and INtime both execute in their own hardware tasks, which means they have their own execution environment and memory mapping. The INtime

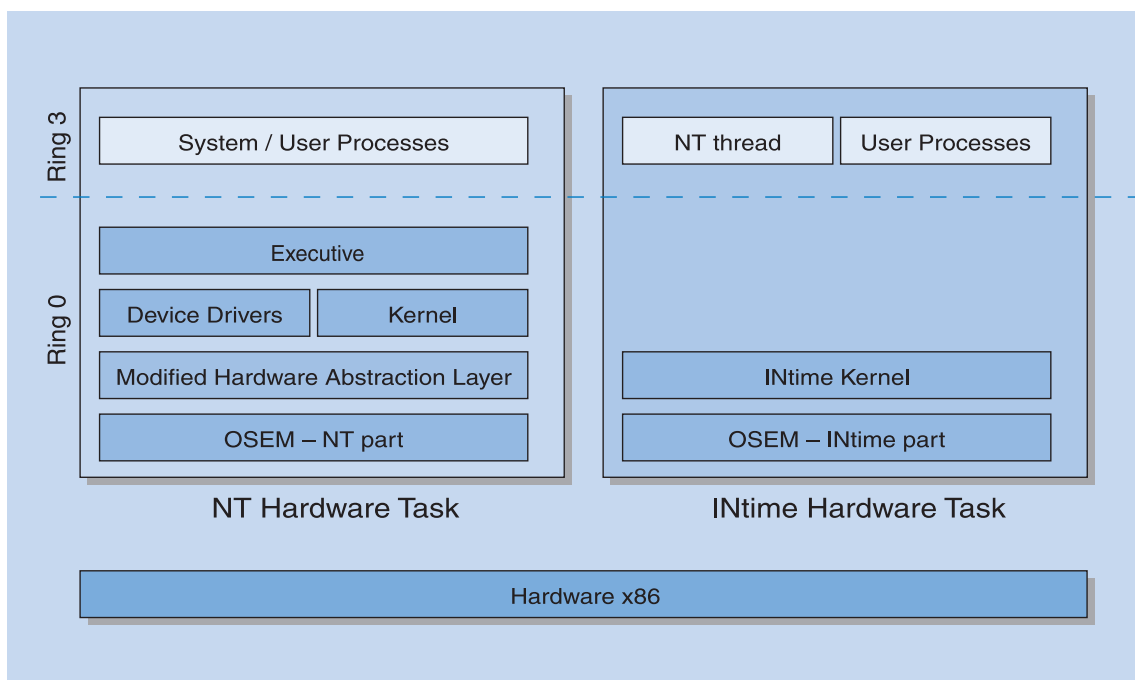


Figure 1. INtime 1.20 system architecture

RTOS EVALUATIONS

system is protected from NT crashes and vice versa. The drawback is in the time needed to switch from Windows NT to the INtime system, adding additional latency when an INtime thread becomes ready. For more details on these added latencies, the reader is referred to the full evaluation report.

The Windows NT hardware task is represented by the lowest priority thread in the INtime system i.e., NT will only get a chance to run when no INtime thread is ready to run. Modifications have been made to the Windows NT Hardware Abstraction Layer (HAL) to guarantee prioritized handling of INtime interrupts over Windows NT interrupts.

Tasks

INtime 1.20 is a multi-process, multi-thread kernel. Priority based pre-emption is used for scheduling threads. A round-robin scheduling policy is used for equal-priority threads.

Processes however can not be dynamically created. This is an application segment limitation that renders INtime less appropriate for the use in particular types of applications. A more detailed discussion on this can be found in the evaluation report.

INTIME 1.20	
Model	Threads and Processes
Priority levels	255
Max. nr of tasks	Limited by the amount of available memory
Scheduling policies	Prioritized round-robin: 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 • its time quantum expires
Number of documented states	5

Table 1. INtime 1.20 Task handling properties

Memory

The memory used by INtime 1.20 is allocated from Windows NT's non-paged pool during the boot process. This memory pool is a continuous space of physical memory assigned to INtime's root process during initialization. Each INtime process gets its own virtual memory space using memory from the root process, making the system more robust. This also adds protection to the kernel as it runs at privilege level (ring) 0 whereas the applications run at ring 3 (see Figure 1).

(1) The page frame size in an 80386 is 4 KB.

INTIME 1.20	
MMU	Required
Physical page size¹	4KB
Paging/Swapping	No swapping
Virtual memory	Yes
Memory protection models	Private memory protection. Real-time applications run in a different environment than NT applications. Moreover, they run at protection ring 3, protecting the INtime kernel.

Table 2. INtime 1.20 Memory Management properties

Interrupts

INtime 1.20 uses the thread interrupt model. The system installs an Interrupt Service Routine (ISR) for each hardware interrupt line, replacing NT's ISRs.

The interrupt handler installed by the user is an Interrupt Service Thread (IST). The IST is a routine that executes at thread level. The IST for a particular interrupt executes in the context of the interrupted thread. Moreover, the IST inherits the priority of the interrupted thread, which usually cannot be predicted. Under certain conditions this can cause problems, as is demonstrated in the evaluation report.

INtime interrupts (i.e. interrupts handled by INtime) always have priority over Windows NT interrupts.

INTIME 1.20	
Handling	Not nested, not prioritized
Context	Interrupt handler uses context of interrupt INtime thread. When NT runs, the interrupted thread is the lowest priority thread in INtime.
Stack	?
Interrupt-to-task communication	Yes
Min. RAM	?

Table 3. INtime 1.20 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 INtime 1.20. Table 4 gives an overview of all the categories and the score (in percentage points) that was obtained. The full

RTOS EVALUATIONS

evaluation report contains a breakdown of the categories into individual features and system calls.

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

MECHANISM	RICHNESS
Thread Management	41 %
Clock	14 %
Interval Timer	33 %
Fixed block size memory partition	0 %
Non-fixed block size memory pool	38 %
Interrupt Handling	75 %
Counting Semaphore	80 %
Binary Semaphore	0 %
Mutex	50 %
Conditional Variable	0 %
Event Flags	0 %
POSIX Signals	0 %
Message Queue	44 %
Mailbox	0 %
AVERAGE PERCENTAGE	31 %

Table 4. API Richness

An average percentage of 31% was obtained. The average percentage does not include any weight factors, it is simply the average of each category's score. For the sake of comparison, the average percentage score obtained for RTX 4.2 from VenturCom Inc was 30%.

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 the interrupt handler to the next task scheduled to run.

We would like to point out to the reader that 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 INtime subsystem to

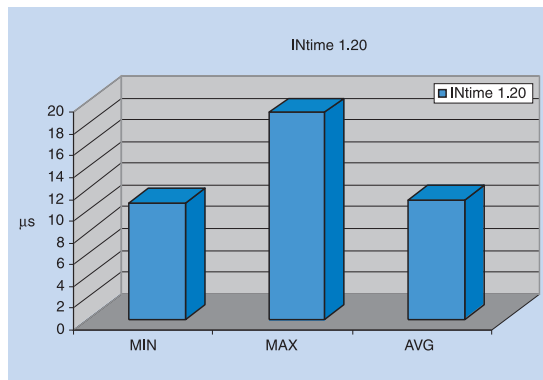


Figure 2. Interrupt Latency - INtime 1.20

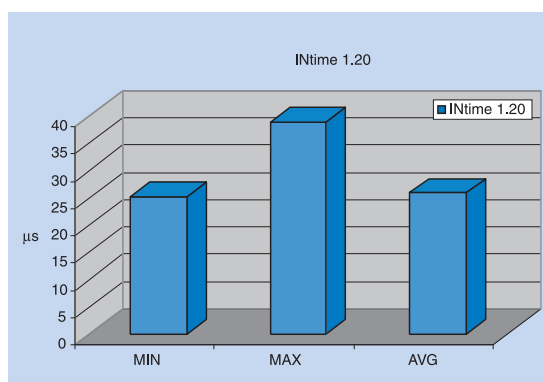


Figure 3. Interrupt Dispatch Latency - INtime 1.20

let the interrupt handler execute. As mentioned earlier, due to the hardware tasking mechanism, switching from Windows NT to INtime (or vice versa) adds some latency to our metric.

As can be seen in Figure 2 and Figure 3, the results are rather slow. This is mostly due to the protection between the user applications (including the IST) and the kernel. More details and other interrupt handling performance results can be found in the evaluation report.

Priority inheritance

INtime 1.20 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

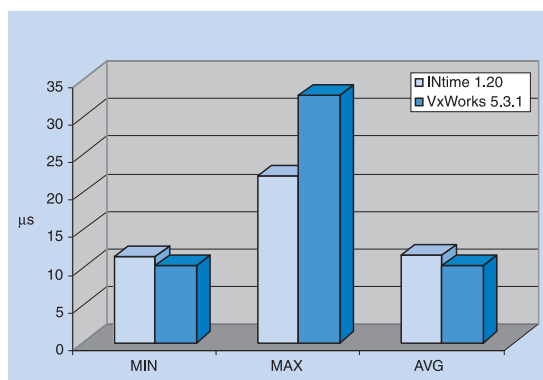


Figure 4. Priority Inheritance

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 from our website (<http://www.realtime-info.be/eval>).

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 test results for VxWorks 5.3.1 from WindRiver Systems were included for the sake of comparison.

TOOLS & DEVELOPMENT METHOD

An INtime application can be developed in two ways. One is the straightforward way as for every other RTOS. The second way is to use bare NT (Win32) to already start developing. However, it is cumbersome to move the Win32 application to INtime due to the change of all system call names. Furthermore, not all the functionality of the INtime API is available from Win32 processes which makes it very hard to prototype the real-time application in Win32.

Two wizards are available to help the developer: one for device drivers and one for INtime applications. However, we observed that the generated code is presented in a chaotic way. Adding personal code to the code generated by the wizard is not straight forward.

DOCUMENTATION & SUPPORT

The documentation gives a good overview but is not complete. The documentation on the system architecture is not very clear and sometimes even confusing. The API system calls are well documented, which is very important to application developers. Technical support is good.

CONCLUSION

INtime's implementation is based on the hardware tasking mechanism provided by the Intel architecture. This approach makes that INtime 1.20 has an execution environment and memory space totally separated and protected from Windows NT. Real-time applications are protected from Windows NT crashes and vice versa.

Within the INtime subsystem, every application process has its own private virtual memory space. Moreover, since applications execute in ring 3, privilege protection is provided between the applications and the kernel, which executes in ring 0. All this makes INtime 1.20 a robust and secure system.

INtime 1.20 exhibited predictable behavior during our tests, but its complex architecture reduces performance somewhat.

Due to the application segment limitation described earlier, INtime 1.20 is not appropriate for the use in certain real-time systems.

A comparison report is commercially available on our website (www.realtime-info.be/eval) for readers who are interested in how INtime 1.20 is situated against other Windows NT real-time extension products.

OTHER PUBLICATIONS AND SERVICES

As mentioned at the outset, evaluation reports for Windows NT 4.0, RTX 4.2, Hyperkernel 4.3 and INtime 1.20 have been commercially available on our website since the beginning of 1999.

Real-Time Consult has also evaluated VxWorks/x86 5.3.1, QNX 4.25 and pSOSystem/x86 2.2.6. Evaluation reports for these products have been available since April 20, 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 performs feasibility studies and product validations on customer demand. Please contact our offices for additional information. ■

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.