

Hyperkernel 4.3 Evaluation Executive Summary

The following article is the executive summary report of the evaluation of Hyperkernel 4.3, the real-time extension to Windows NT from Imagination Systems Inc. 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.
- pSOSSystem/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 Hyperkernel 4.3 from Imagination Systems Inc.

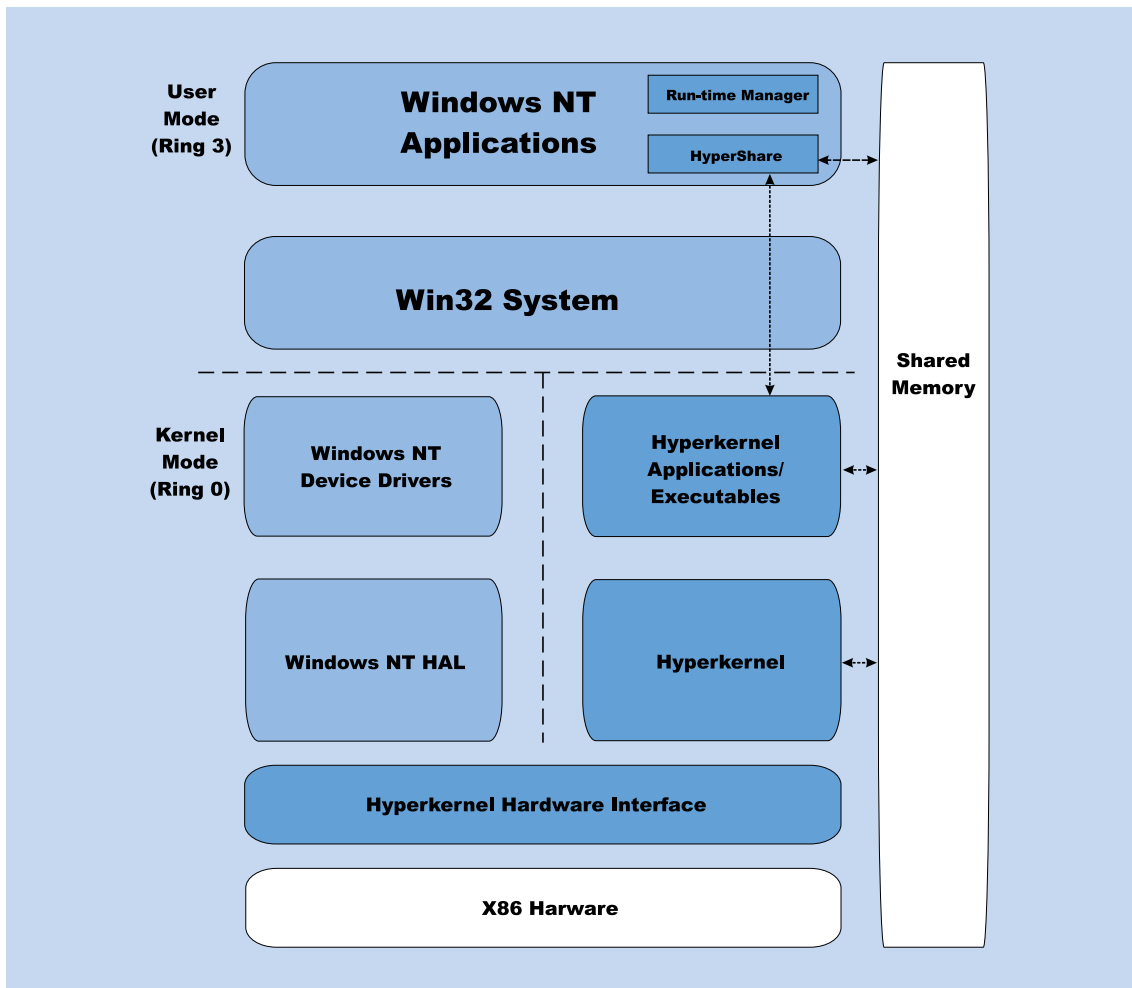


Figure 1. Hyperkernel 4.3 System Architecture

ARCHITECTURE

Hyperkernel 4.3 is implemented as a self-contained execution environment with its own scheduler, its own set of services and its own kernel. Figure 1 illustrates how Hyperkernel 4.3 works in conjunction with NT.

A general approach to make the native NT part coexist with the real-time part is to treat NT as a single task with the lowest priority in the real time part. The problem with this approach is that it might easily lead to "NT starvation", i.e. the NT part does not get enough time to run and crashes.

Imagination System has tried to deal with this shortcoming by alternating the execution of Windows NT and Hyperkernel. A time-slice (duration is configurable by the user) determines how long Hyperkernel can run without getting pre-empted by Windows NT, or vice versa. This mechanism has some serious consequences, since it is possible for any action in the real-time application to be pre-empted to let (less time-critical) Windows NT run. It is demonstrated in the full evaluation report how this mechanism makes the execution of the application almost unpredictable.

Hyperkernel 4.3 does have system calls to create a section in the code that is guaranteed to execute without NT intervention. However, these sections can only be created in carefully selected areas since they completely shut out Windows NT, even if the real-time application becomes idle. The usefulness of these sections is therefore limited.

Tasks

Hyperkernel 4.3 uses a thread-only model. Objects called "processes" are also supported, but they are actually threads since all "processes" share one common address space. The only difference is that a "process" is loaded from a separately compiled and linked executable file. Internally, "processes" are handled like threads.

HYPERKERNEL 4.3	
Model	Threads
Priority levels	32
Max. nr of tasks	1024
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. Hyperkernel 4.3 Task handling properties

Memory

Hyperkernel uses one flat memory space. Hyperkernel processes are not protected from each other. However, NT and Hyperkernel 4.3 are protected from each other by the memory assigned to them, i.e. NT cannot affect Hyperkernel memory space and vice-versa. In Hyperkernel 4.3, the kernel and all user applications execute in ring 0.

HYPERKERNEL 4.3	
MMU	Required
Physical page size¹	4KB
Paging/Swapping	No
Virtual memory	Not supported
Memory protection models	No protection, all system and user threads share one common address space

Table 2. Hyperkernel 4.3 Memory Management properties

Interrupts

It is not completely clear how interrupts are handled in Hyperkernel 4.3. The documentation misleads the reader by insinuating that the interrupt handler installed by the user is an actual ISR (Interrupt Service Routine). Our tests however proved that this interrupt handler is instead a function running at thread level. Hyperkernel 4.3 uses a thread interrupt model, and the interrupt handler installed by the user is an Interrupt Service Thread (IST).

The interrupt handling is unpredictable: the IST inherits the priority of the thread that is interrupted (which is usually impossible to predict) and can therefore be pre-empted by other threads delaying the execution of the IST indefinitely. The system calls to disable scheduling within Hyperkernel are no solution to this problem as they cannot be executed from within an IST. For a more detailed discussion, the reader is referred to the full evaluation report.

HYPERKERNEL 4.3	
Handling	Not nested, not prioritized
Context	Interrupt handler uses context of interrupt thread
Stack	Interrupt handler uses stack of interrupt thread
Interrupt-to-task communication	Yes
Min. RAM	Less than 50 bytes

Table 3. Hyperkernel 4.3 Interrupt handling properties

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

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 Hyperkernel 4.3. Table 4 gives an overview of all the categories and the score (in percentage points) that was obtained. The full evaluation report contains a breakdown of the categories into individual features and system calls.

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

MECHANISM	RICHNESS
Thread Management	41 %
Clock	29 %
Interval Timer	33 %
Fixed block size memory partition	0 %
Non-fixed block size memory pool	15 %
Interrupt Handling	38 %
Counting Semaphore	0 %
Binary Semaphore	0 %
Mutex	33 %
Conditional Variable	0 %
Event Flags	0 %
POSIX Signals	0 %
Message Queue	19 %
Mailbox	0 %
AVERAGE PERCENTAGE	15 %

Table 4. API Richness

An average percentage of only 15% was obtained. RTX 4.2, a real-time extension from VenturCom Inc, scored an average of 30%. Hyperkernel's API is limited, and several basic objects are missing. It is worth mentioning that Hyperkernel 4.3 does have objects called "semaphores", but this name is misleading. The same thread that acquired the "semaphore" must release it. This is the behavior of a mutex.

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.

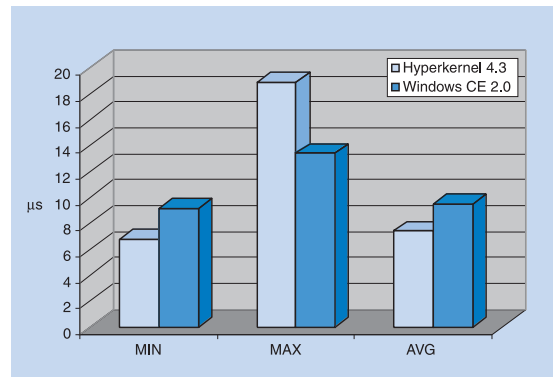


Figure 2. Interrupt Latency – Hyperkernel 4.3

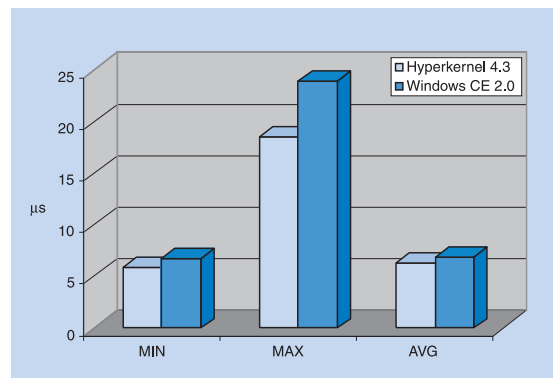


Figure 3. Interrupt Dispatch Latency – Hyperkernel 4.3

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 Hyperkernel sub-system to let the interrupt handler execute.

As described earlier, Hyperkernel 4.3 uses a thread interrupt model: the interrupt handler (IST) is a function running at thread level. The results for Windows CE 2.0, which also uses a thread interrupt model, are included for reference.

No Priority Inheritance

Hyperkernel 4.3 provides a mutex (the Hyperkernel manual refers to it as a semaphore, which is somewhat misleading). However, no mechanism to avoid priority inversion is provided.

A mutex in Hyperkernel 4.3 has an unusual implementation: as opposed to being an object, the mutex behavior is simulated with a thread. The mutex-thread's priority is entered by the user when the mutex is created. This mechanism can easily cause the priority inversion problem to occur, as discussed in the full evaluation report.

Another consequence of this mechanism is the reduction of speed since a context switch is required every time the mutex is acquired or released. This is clearly

RTOS EVALUATIONS

demonstrated in Figure 4, which compares uncontested mutex acquisition times for Hyperkernel 4.3 and RTX 4.2 real-time extension from VenturCom Inc. RTX 4.2 has a conventional mutex-object implementation.

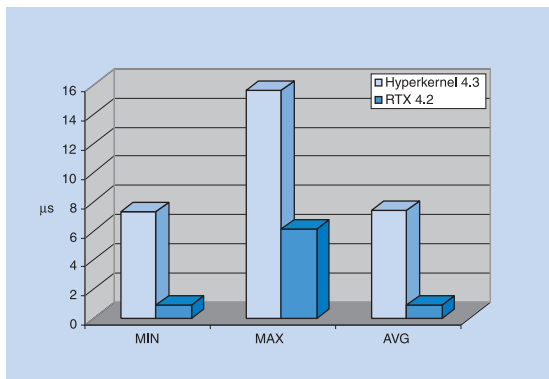


Figure 4. Mutex acquisition time

TOOLS & DEVELOPMENT METHOD

As with the other real-time extensions to Windows NT, a Hyperkernel application can also first be prototyped as a Win32 application. However, not all the functionality of the Hyperkernel API is available in Win32, complicating the development of the Win32 prototype.

Porting the Win32 prototype to the Hyperkernel subsystem is manageable since the Hyperkernel API is based on the Win32 API. The real-time applications execute at kernel level, meaning that developers have to resort to kernel-level debugging tools. Debugging kernel level applications is substantially different from debugging conventional applications.

DOCUMENTATION & SUPPORT

The documentation overall is very poor and misleading in certain cases. The section on the API system calls does not provide a detailed enough description of the system call parameters, which might lead to misunderstandings. The manual mentions very little on architectural issues. We did enjoy good technical support during our evaluation.

CONCLUSION

Hyperkernel is implemented as its own self-contained execution environment. In an attempt to prevent Windows NT from "starving", Imagination Systems opted for a approach to let Windows NT and Hyperkernel run alternate. However, this time-slice mechanism makes the execution of the real-time application almost unpredictable.

As mentioned earlier in this summary, Hyperkernel 4.3 has some other predictability issues, especially concerning interrupt handling and mutexes. These issues are described in great detail in the full evaluation report.

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.