

# Can Windows NT 4.0 be used as an RTOS ?

When Windows NT was originally designed it was not foreseen to be a real-time operating system. Yet many people are keen to use it as such and some already do so. Why would they want to do this and what are the advantages of Windows NT? This paper looks into those questions and explains the general architecture of the operating system. It also gives some results of performance tests that are useful for anyone who's interested in using Windows NT in real-time systems.

## REAL-TIME

This investigation is into the real-time characteristics of operating systems. For an operating system to be classified as a good real-time system the following elements are required:

- The operating system needs to be multithreaded and pre-emptible;
- The notion of thread priority has to exist;
- The operating system has to support predictable thread synchronization mechanisms;
- A mechanism for avoiding priority inversion must be available.
- The OS behavior should be known and predictable (interrupt latencies, task switch latencies, driver latencies, etc). This means that there should be a maximum response time under all load circumstances.

It is important to note that these requirements are necessary but not sufficient.

## ARCHITECTURE

Windows NT is a relatively new operating system. It was designed in 1989 from the start as being a modern operating system for the nineties and beyond. The basic idea rested on object-oriented programming, allowing the operating system to be built in modules. This idea of abstraction was also extended to the kernel so that the operating system would function independently of the underlying hardware. This was achieved using a Hardware Abstraction Layer, the HAL.

### HAL

Following the object-oriented model all components of the operating system communicate by sending messages to each other. Calls to system services are realised by a message being sent from the user mode process to the kernel mode service. This allows each part of the operating system to be developed independently and increases the overall reliability of the design. Applications do not have direct access to the hardware as this is managed by services running at

kernel level. This allows NT to be easily ported to different platforms. Regardless of the platform the structure discussed here remains the same.

### Task Management

Processes in Windows NT belong to two different priority classes: dynamic or real-time. Most processes belong to the dynamic class, which allows the operating system to change their priority depending upon factors such as whether they are a foreground task or they have been idle recently. This is good for a GPOS as it gives all threads a chance to run and gives the user quicker reactions from the active application. However, the rules determining these priority changes are not suitable for an RTOS so Microsoft included a range of priorities above the dynamic class called the Real-Time Class (Figure 1).

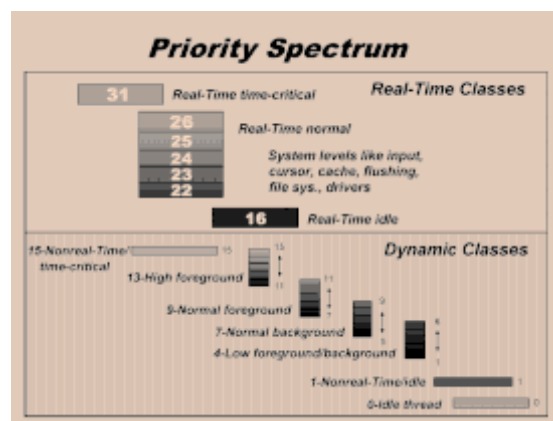


Figure 1 Windows NT priority spectrum (from [1])

Threads running at these priorities always execute before those in the dynamic class. The operating system does not change their priorities either, thus leaving more control to the developer. Microsoft does not recommend that threads spend a long time in this class as they have priority over some system administration tasks such as disk cache flushing and input control.

One might have thought that this would be enough to

# RTOS EVALUATIONS

make Windows NT an RTOS. Unfortunately, there are some problems with their solution. There are only 7 priority levels that can be used within this class, which are not enough to make a serious real-time application.

A deadlock of priority inversion can occur on threads waiting on a mutex. Most RTOSs solve this problem with a form of priority inheritance but Windows NT uses a scheme where threads that have not run for some time receive a random priority boost enabling them to run. This is not predictable and so is unacceptable for a real-time system but it is usable in a general-purpose operating system. Further details can be found in [3].

## Memory

Virtual memory management was included from the start and each application process runs in its own memory space. This increases reliability by preventing processes from corrupting others. But as it isolates tasks from the hardware it is much more difficult to access it directly. Just reading from PCI memory for example requires a device driver to be written to map the desired PCI memory space into the application's address space.

The virtual memory management includes a swapping mechanism. For business applications, this is great, but for a real-time system that has to respond to external events in a predefined time limit, this generates unpredictability when the system has to get a memory page from disk. Therefore, Windows NT allows you to lock pages into memory through the call to the VirtualLock function. In his book, Jeffrey Richter says that Windows NT can still unlock the pages and swap them out to the disk if the whole process is inactive [6]. However, we have not managed to demonstrate this. The only thing we manage to find that the memory allocated for the window can still be swapped out when you iconise them. So normally for real-time applications, this will not be a problem. Is the previous point still a problem? It is not, if the application and other third party applications are well designed and do not take more memory than physically available.

At the device driver level, however, swapping can be avoided.

## Interrupt Handling Method

A real-time system interacts with the external world via the computer hardware. External events are converted into interrupts and handled by a device driver. The device driver is responsible for handling the interrupt generated by the hardware it controls. To do this, an original mechanism has been chosen in order to increase the responsiveness of the OS. The interrupts are handled in two stages. First, the interrupt is handled by a very short Interrupt Service Routine (ISR). This will do the minimum to store the contents of hardware registers and acknowledge the interrupt. It then asks for the rest of the work to be done by a DPC (Deferred Procedure Call) which will be executed afterwards [5]. The DPC is placed in a FIFO queue and is then run once the other DPCs before it have finished. There is no notion of priority in the DPC queue so DPCs from lower priority interrupts will be handled before DPCs from higher (but later) priority interrupts. There is no way

of knowing how many DPCs may be queued before and, as they are supplied from third parties, no indication of the time they will take to execute.

Separating the DPC from the ISR reduces the amount of time that interrupts are masked. It allows quick response to hardware for the urgent things such as register reading. But as the DPCs are queued in FIFO order there is no guarantee on the time that it'd take before the DPC is run. This reduces the predictability of the system and hence makes it unsuitable as an RTOS. However the rapid response outweighs the lack of predictability for a GPOS and the test results show that on average the interrupt latency is very fast.

## The Win32 API

All these points seem rather negative for those who are looking for a serious reliable RTOS. But Windows NT does have its advantages and the main one is the Win32 API. This application-programming interface must be the most used one in the world so there are many developers who know and understand it well. It is also very rich with many synchronisation objects. The development tools and integrated development environments are very advanced and easy to use. This simplifies the developer's job greatly and opens the real-time market to new developers and vendors.

## PERFORMANCE

After the technical approach in the first section, this part concentrates on some time measurements. The measurements have been carried out with an Intel Pentium 200Mhz MMX processor on a classical PCI ATX motherboard, with the cache enabled. The hardware specification should be taken into account when reading the results. The x86 platform has been chosen to provide a standard platform for all our evaluations. The figures published here could be compared to other evaluations but here we will consider the differences with QNX Neutrino. Comparisons with Windows CE have also been included although CE is not yet classified as a good real-time operating system.

Measurements were taken with an external hardware timer. Software timers often do not have enough precision and are synchronous with the system clock. During the execution of a test tracing data was written to a valid PCI bus address before and after the event to be measured. Writing a trace takes about 180ns to

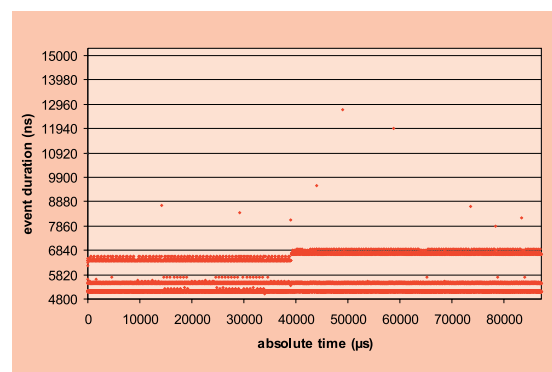


Figure 2 Windows NT 4.0 Workstation - Thread Switch Latency for 10 Threads

## RTOS EVALUATIONS

210ns on the 33MHz bus in the test machine. A PCI analyser was used to store the data written on the PCI bus into local memory and add a timestamp to it. After a test was completed the data was downloaded to a PC for further processing.

### Threads

The first test measures the thread switch latency, i.e. the time to switch from task to task. A different number of tasks were used for this test, from 2 tasks to 128. All the tasks run at the same priority level in the real-time class and voluntarily relinquish the CPU using a system call that puts a task at the end of the priority queue. The function that voluntarily releases the processor is the Sleep(0) command. All the threads belong to the same process.

The detailed information about task switch latencies for 10 tasks are shown in Figure 2. The average value is  $5.487\mu\text{s}$  while the maximum is  $12.720\mu\text{s}$ . These values are quite long when compared to QNX Neutrino as can be seen in Figure 3.

Further results (Figure 4) show that the average thread switch latency as well as the maximum rises with the number of threads in the process. The quite exceptional maximum of 1.2ms was measured in other tests showing that it doesn't just occur when there are 128 threads in the system. This graceful degradation is fine in a GPOS, as the system should continue to run (al-

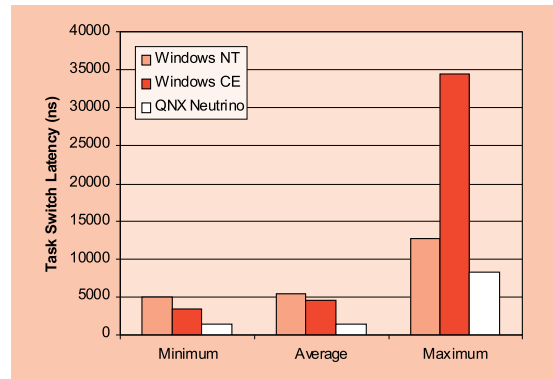


Figure 3 Task Switch Latency between 10 Tasks for Windows NT, Windows CE and QNX Neutrino

Ad  
SBS OR Computers

# RTOS EVALUATIONS

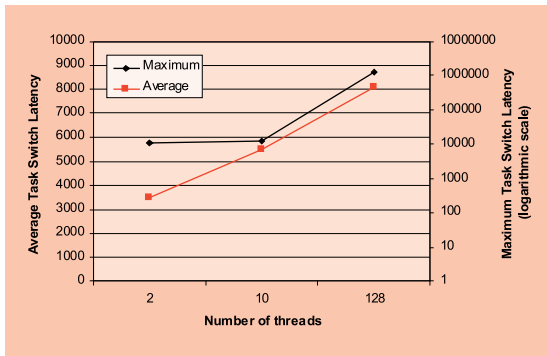


Figure 4 Average and Maximum Task Switch Latencies in Relation to the Number of Threads for Window NT

beit slowly) under increased load. However, in an RTOS, the maximum should not increase with the load and certainly shouldn't be so large compared to the average value. A task switch latency in the order of milliseconds is too slow for most real-time applications.

## Interrupts

The first metric measured by the tests are latencies for interrupt handling. In our test, we implemented a typical RTOS interrupt model: the quick handling of the interrupt is done in an interrupt service routine (ISR). The interrupts are generated by an external device every  $50\mu\text{s}$  ( $\pm 5\mu\text{s}$ ) on the PCI bus. As this device has an onboard clock, the interrupts are generated asynchronously to the motherboard clock. There are two metrics measured by this test: the ISR latency and the ISR dispatch latency.

The ISR latency is the time interval from the last line executed in the interrupted thread to the first line executed in the ISR handling the current interrupt. These values only indicate the time it takes to start handling an interrupt. According to Microsoft the minimum amount of work possible should be done in this ISR, the real handling should be performed in a DPC that would run afterwards. This DPC would be delayed by random amounts of time depending on the other device drivers installed on the machine. As different configurations would have different device drivers our tests do not use a DPC.

The ISR dispatch latency is the time it takes to go back to a task upon leaving an ISR. It is the time between the last line of the ISR to the first line executed in the first task that runs. DPCs have a higher priority than tasks (even tasks in the real-time class) so they would be executed before returning to the task that was interrupted. Again, as these DPCs introduce variable delays, a DPC was not attached to our ISR.

### ISR Latency

The results for ISR latency are shown in Figure 5. The average value is  $3.948\mu\text{s}$  and the maximum peak is  $22.830\mu\text{s}$ . The peak could be caused by another higher-priority interrupt such as the clock delaying ours. The average value is quite fast especially when compared to real-time extensions but in an RTOS it is the maximum value that is important, not so much the average. The maximum value is very high when compared to the average, which would give a large standard deviation. A large standard deviation means that

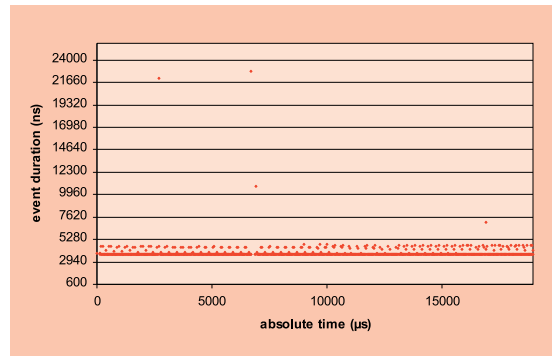


Figure 5 Windows NT 4.0 Workstation - ISR Latency

there are too many values far from the average making the system less predictable. When these results are compared with QNX Neutrino (Figure 6) it can be seen that Windows NT has a long ISR latency and that its maximum is indeed rather too high. This makes NT unsuitable for use as an RTOS but as a GPOS has less need to service interrupts this is less of an issue.

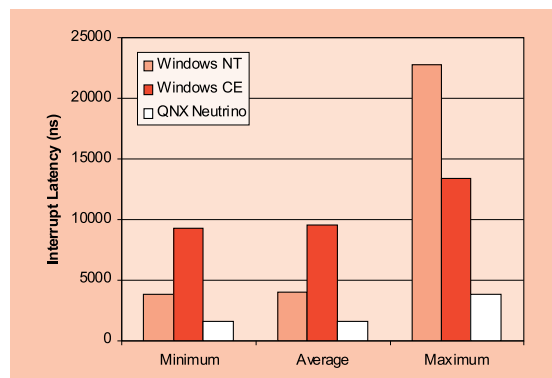


Figure 6 ISR Latency for Windows NT, Windows CE and QNX Neutrino

The results for Windows CE seem rather long but that is because of the way it handled the interrupt. It was not possible to create an ISR with our tracing functions that access mapped memory. The only thing that could be done was to run an Interrupt Service Thread (IST) afterwards that runs in a similar fashion to a DPC. An IST is a task with a very high priority and is not placed in a FIFO queue like a DPC. It will therefore not be delayed by low priority interrupts. Hence the values for Windows CE are the time from the last line of the interrupted task to the first line of IST including the time spent in the ISR that calls the IST. This is the time it takes before the hardware can be handled and so the results can be compared across the different operating systems.

### ISR Dispatch Latency

Figure 7 shows the results for ISR dispatch latency. The average value is  $10.753\mu\text{s}$  and the maximum is  $29.010\mu\text{s}$ . This is the time it takes to get from the ISR back to the interrupted task. If a DPC were used (as should be according to Microsoft guidelines) then this time would be lengthened by the time spent in the DPC and context switches. However we have kept

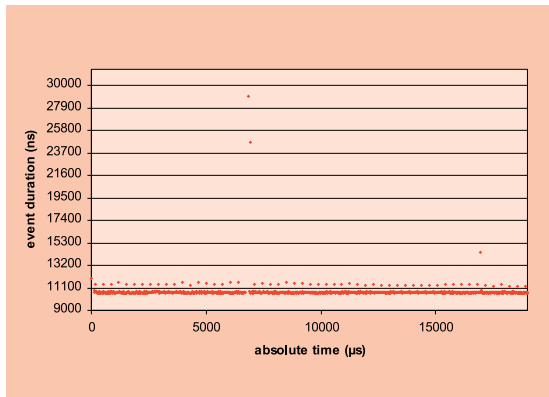


Figure 7 Windows NT 4.0 Workstation - Interrupt Dispatch Latency

these values so that they can be compared with those for QNX Neutrino (and other evaluations). The ISR dispatch latencies for NT are very long when seen in the context of what QNX achieves and may be too long for most real-time applications. But, again, as GPOSs have less need to handle interrupts quickly these values are acceptable for using Windows NT as a GPOS.

## CONCLUSION

Windows NT has fast average ISR latencies and thread switching latencies. This is an important metric for general-purpose operating systems. The stability and reliability of the architecture as well as the immense richness of not only the API but also the range of programs available makes Windows NT a very good choice as a GPOS. However, the exceptionally high maximum values are likely to cause problems for all but the least demanding real-time applications. It is to avoid these high values that a whole market has emerged to develop and sell real-time extensions to NT to improve the predictability of the operating system. These vendors obviously believe that NT alone is not enough for real-time systems. People may continue to use Windows NT for their real-time solutions; one can only hope that they have taken into account the millisecond delays.

The complete RTOS Evaluation Report of Windows NT is available from of November 12 1998 at <http://www.realtime-info.be>. ■

## REFERENCES

- [1] M. Timmerman, B. Van Beneden & L. Uhres, RTOS Evaluations Kick Off, Real-Time Magazine, Issue 98-3 (<http://www.realtime-info.be>).
- [2] Technology Brief: Real-Time Systems with Microsoft Windows NT, Publication of the Microsoft Developer Network, Microsoft Corporation, 1995.
- [3] M. Timmerman & J.C. Monfret, Windows NT as a real-time OS, Real-Time Magazine, Issue 1997-2 (<http://www.realtime-info.be>).
- [4] David A Solomon, Inside Windows NT Second Edition, Microsoft Press, 1998.
- [5] Microsoft® Windows NT™ Version 4.0 Device Driver Kit Documentation.

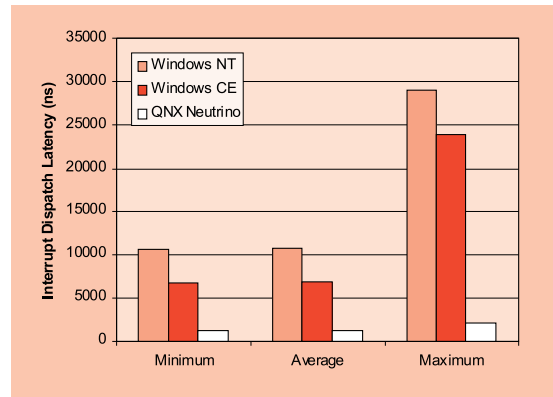


Figure 8 Interrupt Dispatch Latency for Windows NT, Windows CE and QNX Neutrino

- [6] Jeffrey Richter, Advanced Windows, The Developer's Guide to the Win32 API for Windows NT 3.5 and Windows 95, Microsoft Press, 1995.

*Dr. Ir. Martin Timmerman graduated in Telecommunications Engineering from the Royal Military Academy (RMA) Brussels and received his Doctorate in Applied Science from the Gent State University (1982). He became the director of the System Development Center (SDC) at RMA, which he created in 1983, and converted himself to a Computer Science Engineer. Presently, he gives general courses on Computer Platforms and more specific courses on System Development Methodologies at the RMA. Outside the RMA, Martin is known for his audits, reviews, seminars, evaluation reports and feasibility studies he performs with Real-Time Consult. Real-Time Consult is also known for Real-Time Magazine and the Real-Time Encyclopaedia web-site (<http://www.realtime-info.be>) His second company, Real-Time User Support International (RTUSI) 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 1997 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.*

*David Newman obtained his degree in Computer Systems Engineering from the University of Bristol, UK in 1996. After travelling he joined Real-Time Consult at the start of 1997 to work as a software engineer on various consultancy projects. David now concentrates his efforts on the RTOS Evaluation program of Real-Time Magazine.*

\* Although all care has been taken to obtain correct information and accurate test results, Real-Time Consult and Real-Time Magazine cannot be liable for any incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report, even if Real-Time Consult and Real-Time Magazine have been advised of the possibility of such damages.