By Martin Timmerman, Chief-Editor of Real-Time Magazine,
Bart Van Beneden Project Manager,
Laurent Uhres, Software Engineer,
Real-Time Consult.

Is Windows CE 2.0 a real threat to the RTOS World ?

This article gives an overview of the capabilities of Windows CE 2.0 as a real-time operating system (RTOS). This article is part of an evaluation project for real-time operating systems.

INTRODUCTION

The first version of Windows CE was released in November 1996 and it was intended for hand-held PCs. Version 2.0 was introduced a year later at the end of 1997. Some changes have been made to the operating system since its first release to address problems experienced by the users of version 1.0, to increase its performance and to reduce the size of the executive.

With the introduction of Windows CE 2.0, Microsoft wants to gain market share in the embedded operating systems market. It clearly marks Microsoft's attempt to enter the embedded systems market. Microsoft understands the need of an operating system to support real-time to enter this type of market. In this article, we discuss the potential use of Windows CE 2.0 for embedded real-time application development. Statements on the use of Windows CE 2.0 for such systems have been made in several articles released by Microsoft, including [1], [4] and [5]. In [4], Mark D. Peterson, project manager of Windows CE OS & Tools Division Microsoft, states "Windows CE 2.0: The Real-Time Embedded OS". In most articles released by Microsoft these kind of statements are used as sales arguments without being really validated by technical data. Except [1], which gives measures of latencies. We will discuss these figures later on. In discussion groups on the real-time and embedded systems (comp.real-time and comp.arch.embedded) Windows CE is generally not accepted as a real-time embedded operating system. We wanted to get to the heart of the polemic and examine the operating system from every angle to give an objective conclusion. This article is an executive summary of the full evaluation report available later this year.

In the next two sections, we concentrate on the requirements for embedded real-time applications and determine whether Windows CE 2.0 can meet them. Then we will comment some results of tests performed by us and compare them to the results of QNX Neutrino 1.0.

EMBEDDED

In this first section we focus on the embedded attribute of the application we would like to develop for Windows CE 2.0. The most important factor for embedded applications is their memory size. Even if the price of memory lowers every day, a small memory footprint reduces the costs when producing the custom system in large quantities. The key to success for an operating system is to have a modular architecture as opposed to a monolithic architecture. Modular systems have the advantage of being adjustable to contain only the required functionality.

Modularity

Microsoft insists on the modularity of the new version of its 32-bit operating system. This modularity allows the developer to include only the required modules for his application. The benefit of modularity is to reduce the size of the operating so as to contain only the required functionality. It also reduces the complexity of a system. A high number of modules increases the interaction amongst them, which may in some cases cause system failures such as deadline misses and deadlocks. Furthermore modularity allows the developer to replace specific modules with proprietary or third party modules. Microsoft has provided the operating system with a considerable set of modules, which includes support for FAT file system, Java and a TCP/IP protocol stack. Some of these modules are even subdivided into smaller components. However the minimal configuration requires 330kbytes of ROM and 28kbytes of RAM (see [3]). A different paper published by Microsoft ([2]) states that "a single non-display application would require less than 500K of ROM and 350K of RAM". This is a lot compared to other real-time operating systems like QNX's Neutrino 1.0, which requires less then 128Kbytes for the equivalent minimal configuration.

Another point is that Windows CE is completely ROMable. This means that the operating system can execute from ROM with the use of some RAM for the registry, the Windows CE database, the heaps and the stack.

Portability of Windows CE 2.0

The system architecture has been designed to easily port the operating system to a custom platform. Most of the platform-specific code resides in the OEM Abstraction Layer (OAL). This layer is situated between the kernel and the platform hardware (see Figure 1). Microsoft supplies a tool called OAL Adaption KIT (OAK) to port the OAL to a specific target platform.

Some information and guidelines are given in the manual to proceed. The CPU requirements are listed in the manual. An e-mail address is provided to get information on specific processor support. But the documentation is not as complete on the procedure to port the OAL and to build a new kernel as one would expect.

Application Programmer Interface

The CE application programmer interface (API) is a subset of the Win32 API. Approximately half of the interfaces routines of the Windows NT API are implemented. This makes a total of about 600 routines. People developing for NT are used to this interface and will be able to easily implement programs. When developing applications for CE, they can keep on using the Windows integrated development environment (IDE). But when developing the OAL the developer has to get back to the basic DOS command prompt.

As the attempt to standardise the API for real-time operating systems has not been as successful as excepted, the Win32 API could rapidly become a de facto standard because of the omnipresence of NT and Windows95/98.

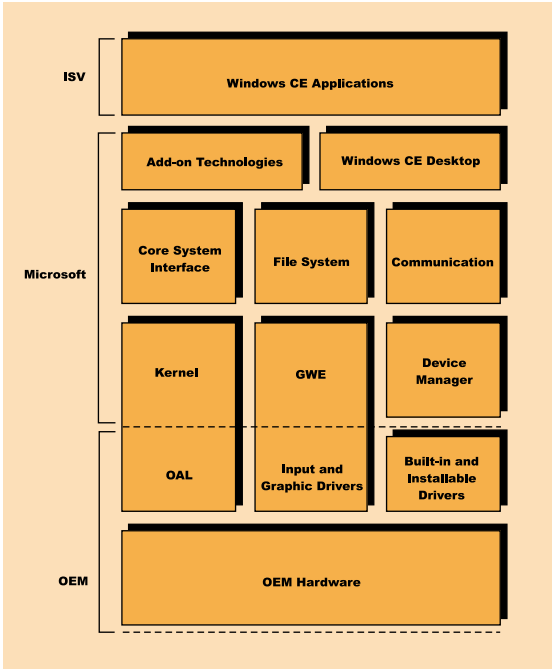

Figure 1 System architecture

REAL-TIME

The second point of our investigation is the real-time characteristic of the application. For an operating system to support real-time 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 priority inheritance 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.

The Kernel: An Overview

Before studying this topic, we need to have an overview of the kernel functionality. Figure 1 shows the architecture of CE. The kernel resides above the OAL layer described in the previous section. It includes memory management, process management and exception handling.

Multitasking and multithreading is implemented by a pre-emptive priority-based scheduler using time-slicing within a priority level (except at the highest priority level). Priority levels assigned to threads are static. A system of priority inheritance exists that adjust thread priority levels to prevent priority inversion mechanism.

The system uses the MMU for virtual memory management. A single 4Gbyte virtual address space is created by the operating system. Although all the processes run in this common memory, they are protected from each other because the kernel assigns to each process a unique slot in memory.

Interrupts are handled by the system in a traditional way: when an interrupt occurs, the kernel looks up the IRQ and calls the registered interrupt service routine (ISR). An important point is that interrupts can not be nested.

Interrupt Handling

The fact that interrupts can not be nested means that all interrupts are disabled during the processing of an interrupt. Especially, high level interrupts are not taken into account when a low-level interrupt is processed. This adds latencies to the system that are difficult to predict and estimate. The system looses its determinism. Application can miss deadlines because device driver interrupts block your high priority interrupts. What happens is that the execution of a critical application depends on third party elements even when these use low-level interrupts. A limited solution is to minimise the processing time of an ISR. The interrupt model proposed by Microsoft (see Figure) encourages handling the interrupt at task level within an interrupt service thread (IST). Interrupts and ISTs are linked together through an event object. When an interrupt occurs, the associated ISR performs minimal processing and returns an interrupt ID to the kernel. The kernel sets the event associated with the interrupt ID. The IST waiting on that event is notified of an interrupt when the system releases this event. It then starts performing the interrupt handling. Generally ISTs execute at the highest priority level.

Processes and Threads Model

Here we stumble on the second problem: only eight priority levels are available for threads. This is far too few because it is difficult to use methodologies to

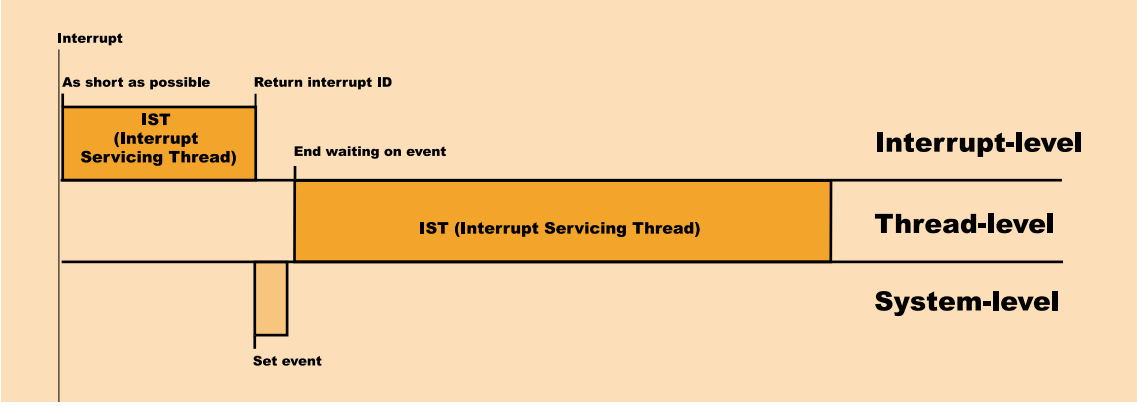

Figure 2 Interrupt handling

design the thread model (e.g. rate-monotonic scheduling). If the system needs more than eight threads, some of them need to run on the same priority level and thus share their CPU time because of the time-slice algorithm. Only at the highest priority level the threads are not time-sliced. They execute until they voluntary relinquish the CPU or when they perform a blocking system call. This level is used for interrupt handling. This can lead to very dangerous situations in a system that needs to handle several different interrupts. Imagine the following scenario where the two different ISTs execute at the highest level: a low-level interrupt occurs and the kernel wakes the associated IST up. While the IST executes a high level interrupt occurs. The system releases the corresponding IST. But this thread will not execute before the low-level handling is completed. The obvious solution is to execute the first IST at a lower priority level. This is applicable in a system with only two different interrupts. In a system with many interrupts, each thread needed to handle an interrupt should be assigned an unique priority level, leaving few levels for the rest of the application. It is easy to see that this low number of priority levels makes it difficult to design a task model and to correctly implement device drivers.

Memory Model

As said before, memory management is based on virtual memory management. This type of architecture adds robustness to an application since they are protected from each other. On the other hand this kind of memory management adds overhead to the execution of an application because references to virtual addresses need to be translates into physical addresses and these addresses need to be checked to ensure the protection.

Windows CE 2.0 features memory mapping allowing threads to share physical memory. Using shared memory allows fast communication between threads. Memory mapping permits also access to memory and registers of hardware devices.

The systems supports on-demand paging, which means that only the page required for the execution of a RAM-based application, is loaded into memory. This paging mechanism adds overhead to the execution of a program because access to a page not located in memory incurs a page fault. Only then will the system load the necessary page. Therefore, the API provides a routine that permits to lock a specified region of a process virtual address space into memory avoiding that subsequent access to that region will provoke a page fault.

PERFORMANCE

After a technical approach in the first two sections, we concentrate in this section 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. Other results published by Microsoft [1] and [2] are based on a SH-3 series microprocessor running at 60Mhz. SH-3 microprocessors run at 60MIPS compared to about 400MIPS for the microprocessor we used in our test configuration. The x86 platform has been chosen to provide a standard platform for all our evaluations. The figures published here should therefore be compared to other evaluations we in this issue performed (see also another article on page 11 in this issue - "Windows NT Real-Time Extensions: better or worse ?") and in the past and to forthcoming evaluations (for more detailed information on our evaluation program see the article "RTOS Evaluations Kick Off!" in this issue on page 6).

Interrupts

The first metric measured by the tests are latencies for interrupt handling. In our test, we implemented the interrupt model suggested by Microsoft: the handling of the interrupt is done at thread level, in an interrupt service thread (IST). The interrupts are generated by an external device every 100µs (+/- 5µs) on the PCI bus. As this device has an onboard clock, the interrupts are generated asynchronously to the motherboard clock.

The metric measured by the test is the IST latency. IST latency is the time interval from the last line executed in the interrupted thread to the first line executed in the IST handling the current interrupt. This interval includes the interrupt latency and the time for the kernel to signal the event used to synchronise the IST. The average for IST latency is 9.5µs and the maximum is 13.4µs. The frequency distribution graph (Figure 4) shows that the results are distributed around the average. The peek observed at 18.8µs (Figure 3) is probably due to occurrence of another interrupt, which has not necessarily a higher level. Figure compares the results for

Windows CE 2.0 to QNX's Neutrino 1.0 (see [6] for the evaluation of Neutrino 1.0) on an identical platform and using the same memory model. The chart indicates clearly that Windows CE 2.0 needs some improvement to be competitive.

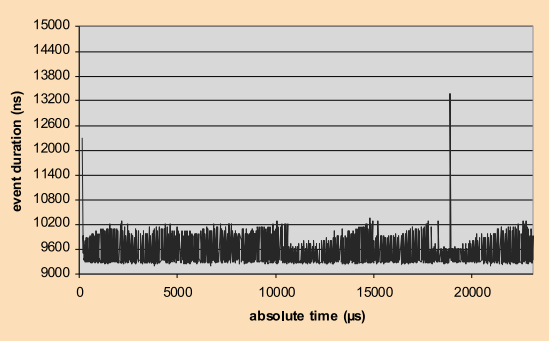

Figure 3. Windows CE 2.0 - IST latency

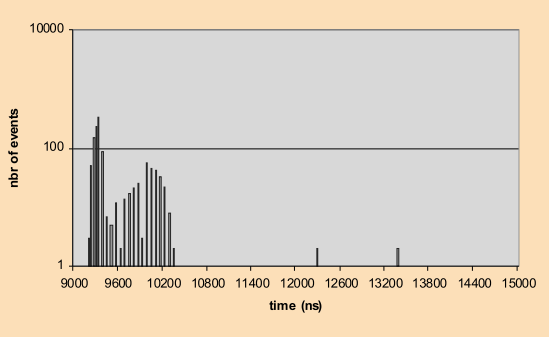

Figure 4. Windows CE 2.0 - IST latency - frequency distribution

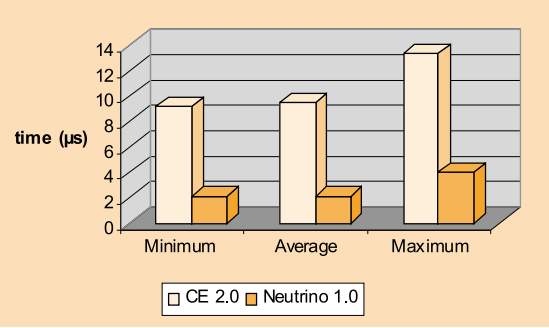

Figure 5. IST latencies comparison

Threads

With the second test we measure the thread switch latency, i.e. the time to switch from task to task. Ten tasks have been used for this test. All the tasks run at the same priority level and voluntarily relinquish the CPU using a system call that puts a task at the end of the priority queue.

The average of the measures (Figure 6) for thread switch latency is 4.4µs and the maximum is 34.4µs.

Figure shows an enlargement of the first 1000µs. Each point on the graph represents the thread switch latency. The first point is the time to switch from first task to the second task. It is interesting to note that the first nine values are higher then the rest (remember that we used ten threads in our test). Windows CE seems not allocate all the necessary resources for a thread to execute when it is created. These resources are allocated when the thread executes for the first time, which explains these initial higher results. Consequently, although the average time for a thread switch is 5µs, the maximum is 34.4µs and occurs at the first execution of the thread. Again, compared to Neutrino (Figure 8), the Windows CE results seem to be higher, especially the maximum values.

Two rules should be applied to a real-time operating system to reduce maximum latencies. Firstly, objects should be ready when created, no additional resource allocation should be done later on. Secondly, system queues should be sorted when objects are added to them, no rearrangement should be done to get an object from a queue. Figure is an example for the first rule. Applying it to Windows CE 2.0 would reduce the maximum value for task switch latency from 34.4µs to 9.4µs. An example for the second rule is the queue of threads pending on an unavailable semaphore. This queue should be sorted on the thread priority level each time a thread is added to it.

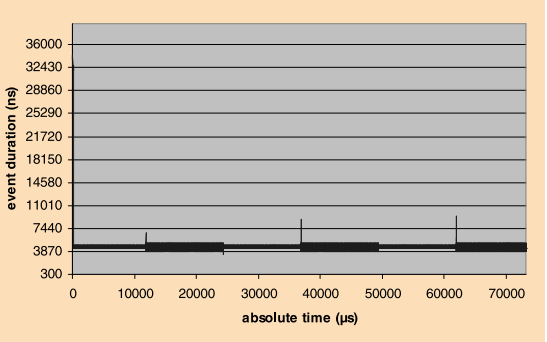

Figure 6. Windows CE 2.0 - Thread switch latency

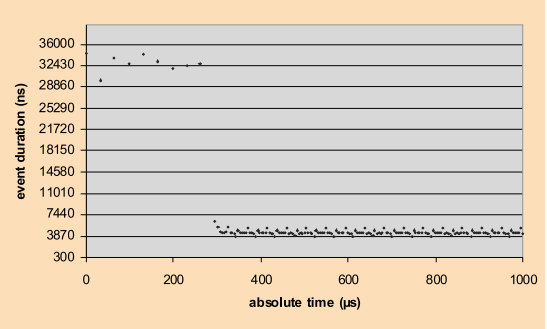

Figure 7 Windows CE 2.0 - Thread switch latency - enlargement

ADDITIONAL THOUGHTS

Finally, it is interesting to note that the article "Using Windows CE for Real-time applications" of the user manual contains the following statements:

"Although Windows CE is not a fully deterministic real-time operating system, its well-designed system of interrupt and thread priorities helps overcome this limitation for real-time applications."

"While the may be certain time-critical (hard real-time) applications for which Windows CE is not appropriate, it does have the ability to handle the majority of today's

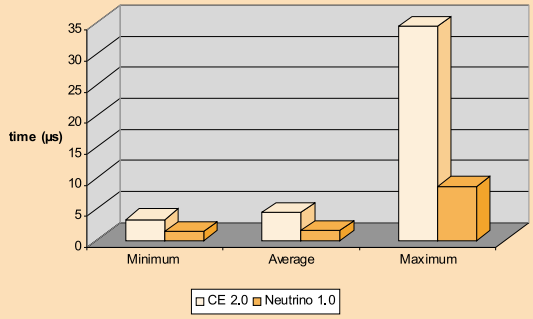

Figure 8. Thread switch latencies comparison

real-time systems."

The first statement contains a contradiction between "not a fully deterministic" and "real-time operating system". A definition of a real-time operating system is that it can handle events in a bounded and deterministic way. Furthermore, we conclude that the system of interrupts is not well designed. The section on interrupt handling explains why. Finally, from what has been said in this article, it is hard to believe the second statement.

CONCLUSION

The conclusions we can draw from this study is that Windows CE 2.0 needs to mature. Performance measurements show that Windows CE 2.0 is not fast enough compared to a similar operating system. The insufficient number of priority levels and the fact that interrupts can not be nested makes the operating system not suitable for medium and large real-time embedded applications. Because for these applications, which handle many interrupts and include a large set of threads, Windows CE 2.0 can not insure deterministic and bounded responses.

This version of CE is more appropriated for small real-time applications like device control and data acquisition although its memory requirements are a little high. These systems have a simple design and low functionality. They require a response time within a 1ms. The number of tasks in such applications is lower then 10. As an example we can think of a system monitoring a small number of temperature probes. The number of probes can hardly exceed two at most three because of the interrupt handling mechanism.

Referring to the profiles defined in [7] based on the POSIX 1003.13 standard, Windows CE 2.0 can be used for small embedded controller systems and embedded dedicated systems.

Its major assets are its modularity and the API based on the Win32 API. Another advantage is the design of the operating system's architecture: it is scalable because of its modularity and it is possible to develop a new board support package.

To overcome the problems pointed out in this article, Microsoft proposes for version 3.0 faster IST latencies, nested interrupts and more priority levels. Let's wait and see...

REFERENCES

[1] "Real-Time Systems with Microsoft Windows CE", http://www.eu.microsoft.com/windowsce/embedded/techpapers/wce20/, 1998.

[2] "Microsoft Windows CE 2.0 - Embedded Solutions", http://www.microsoft.com/windowsce/pie/windowsce20ds.htm, 1998.

[3] "Microsoft Windows CE Memory Use", http://www.eu.microsoft.com/msdn/news/feature/100197/memdrft2.htm, 1997.

[4] Peterson, M. D. , " Windows CE and the Internet ", http://www.microsoft.com/sbnmember/download/cepresentation.ppt.

[5] Pellerin, D., "The Win32 Programming Model: A Primer for Embedded Software Developers", http://www.eu.microsoft.com/windowsce/embedded/techpapers/wce20/, September 1997.

[6] The full evaluation report on QNX Neutrino will be available in October 1998.

[7] Timmerman, M., and Monfret, J.C., "RTOS Benchmark Program", Real-Time Magazine, Issue 95Q2, pag. 12-17.

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.

Laurent Uhres graduated in 1997 as an analyst-programmer from the Haute Ecole Rennequin Sualem, Belgium. He has an additional diploma in Object Oriented Development from the Instito Politécnico do Porto, Portugal. Laurent joined Real-Time Consult mid 1997 to work as a software engineer where he has specialized himself in evaluating RTOSs.