

RTOS Evaluations Kick Off!

Real-Time Consult started a research program about two years ago to qualify and compare real-time operating systems (RTOS). Now, a description and a framework for an evaluation project are defined and ready for use. This paper gives an overview and a framework for that project

INTRODUCTION

These days, software engineers are often asked to develop applications with real-time constraints. They have to choose whether to use an OS (operating system) or a RTOS (real-time operating system). The problem that every developer faces is which RTOS to use as building block for his application, so that he will be able to satisfy the real-time requirements. In order to be able to make a wise choice, tests have to be performed and results of different RTOSs have to be compared with each other. There is clearly a demand to get this kind of information from an independent source.

For quite some time, "benchmarks" have been available for some RTOSs. Most of these benchmarks limit the discussion to measurements of interrupt latencies and task switch latency without any mention of the system's workload. The published figures from different sources are hardly comparable due to a lack of a common test environment.

Therefore we started to deal with this in a more systematic way and defined a framework. We decided not to call it a benchmark project but an RTOS evaluation project, because the objectives are larger than some measurements. Indeed the project started as a Windows OS technology evaluation but today other RTOSs are included.

After two years of experience, doing different feasibility studies for different customers, we came to the following generalized evaluation methodology.

- In a first step, the concept of real time needs to be defined in the context of dedicated systems.
- Next, a definition should be given of what OS features are needed to declare the product a good building block in such environment.
- An evaluation framework has to be put together to be able to rate the quality of an RTOS.

Having defined this evaluation framework, it is used to evaluate various RTOSs.

- We start a theoretical evaluation from the RTOS and vendor documentation to verify that indeed the OS has the (minimum) requirements needed.
- We then perform extensive practical tests that are needed to see if the OS behaves correctly under all load circumstances.
- Once all this is done, then taking into account a defined application, we can then assess whether one or more of these products can be used for this

problem.

This paper has the intention to give an overview of this overall approach.

DEDICATED SYSTEMS WITH REAL-TIME CHARACTERISTICS - TERMINOLOGY

OS technologies are now used in applications ranging from desktops to large dedicated systems. In most publications, people talk about embedded systems. Others are talking about real-time systems, and still others are attaching an enormous importance to the reliability of these systems and talk about fault-tolerant systems. To make things clear, let us have a look at what these terms really mean.

Embedded systems: computer system that is enclosed in another system and makes an essential part of it. Another way of thinking about embedded systems is of hardware and software, which forms a component of some larger system and which is expected to function without human intervention.

Real-time systems: systems that respond in a predictable way to unpredictable external events.

Fault-tolerant systems: systems that continue working under all circumstances (except for physical destruction), or the ability of a system or component to continue normal operation despite the presence of hardware or software faults.

Other definitions are certainly possible but it is impossible to give a definition that will be accepted by everybody. Therefore we consider that it is difficult to keep on working with these expressions - they never convey everything.

Is a cellular phone an embedded system following the definition? Is a car PC a real-time system? All of the today's systems have some aspect of all the different systems. So it doesn't make sense anymore just to talk about embedded or real-time or robust or fault-tolerant. Therefore we started using the term "dedicated systems", having embedded, real-time or fault-tolerant as an attribute.

Dedicated systems: a system where the functionality is once and for all tied up into the hard- and software.

Most dedicated systems have soft real-time constraints, others need to be hard real-time and some have fault-tolerance built in.

MINIMUM RTOS FEATURES

Starting from the definition of real-time given above, dedicated system design engineers need an OS that exhibits predictable behavior. To guarantee this predictability the very first idea would be to have a deadline driven OS, but this technology is not available yet. Therefore the actual RTOS products are priority driven, i.e. the highest priority task ready to run is the one that runs. The developer can then use algorithms like Rate Monotonic Analysis to see if its system would be schedulable under all circumstances. Some tools like PERTS can help him in this task.

The result is that in order to be capable of making the application predictable, the RTOS should meet the following minimum requirements:

- Multi-tasked (or threaded) and pre-emptible priority driven.
- The notion of task (thread) priority has to exist and sufficient priority levels need to be available (depends on the complexity of the application).
- The OS has to support predictable thread synchronization mechanisms (semaphores, mutexes, events, etc).
- A mechanism to prevent priority inversion has to exist.
- The OS behavior (metrics) should be known and predictable (interrupt latencies, task switches, memory latencies, driver latencies, etc.). This means that there should be a maximum response time under all system load circumstances.

EVALUATION FRAMEWORK

It makes no sense to only look for some metrics like task or thread switching times and interrupt latencies. These topics are of course important when it comes to using the product in particular applications. If you want to rate them you should measure them under different system load conditions, as we will explain later. However the choice of an RTOS today should depend on more issues than just metrics.

We created an evaluation framework in which the following items are discussed:

- A short presentation of the product.
- A presentation of the architecture and of its benefits and drawbacks.
- An evaluation of the ease of installation.
- An evaluation of the ease of configuration (to map your application needs).
- Real-Time kernel richness (system calls, priority inversion prevention mechanisms, scheduling policies, etc).
- Richness and performance of the communication mechanisms in general and between NT and the RT part if we discuss an NT extension (synchronous calls, asynchronous calls, and shared memory).
- Kernel performance and predictability (system calls, context switch latencies, task switch latencies, etc.) under different load conditions.

- Standard device drivers performance (TCP/IP, SCSI, Serial driver, etc).
- File system performance and predictability.
- User friendliness (documentation, tools, etc).
- Standard development tools (debugger, profiler, configuration tool, etc).

THE TECHNICAL EVALUATION

The theoretical or technical study, based on product documentation and any other vendor supplied documentation has the following objectives:

- Verifying from a theoretical point of view if the product has no design flaws making it impossible to fulfill the basic requirements of a RT system (if there are none discovered, the practical test should then confirm this).
- Understand the target applications the product is aiming at.
- Describe and analyze the system architecture (monolithic, layered, client-server, etc).
- Make the inventory of available hosts and of supported hardware platforms (processors, I/O devices, BSP, etc).
- Make the inventory of supported languages and tools.
- Study the license policies.

The theoretical evaluation helps in a RTOS selection process to go from a "long list" of candidates to a "short list".

TESTING THE RTOS: PRACTICAL EVALUATION

Overview of the test plan

The practical evaluation presents in-depth information on the RTOS. It answers different questions.

- Is it possible to reduce the size of the kernel to a minimum and to build a system that includes only the necessary components?
- How fast and accurate is the response of the operating system under different workloads?
- What is the overall behavior of the system under (very) high workload? Does the system "gracefully degrade"?
- Eventually, the study can validate the information on the system architecture gathered during a previous technical evaluation.

The practical evaluation evaluates the operating system by different measures. An evaluation of the system's scalability, performance and behavior is included.

One section is a study of the different software components supplied by the vendor. The modules include different system management tasks, the network stack, the file systems and various device drivers. This approach gives information on the scalability of the RTOS and the memory usage for different configurations. It helps to decide whether the RTOS includes the required software components and whether the mem-

RTOS EVALUATION

ory usage of these components is acceptable for the application.

Another section evaluates the performance of the RTOS. The evaluation is based upon a series of tests. The testing measures low-level operations of a RTOS, considering the system under three aspects: task handling, interrupt handling and system calls. The first aspect measures the time needed to switch from one task to the next; the second one measures the reaction time of the system to an external event; and the last one evaluates the performance of the system calls. These system calls include calls to the kernel and calls to device drivers.

The evaluation typically measures specific real-time features of the RTOS. It does not certify that a set of tasks of an application will meet their deadlines. As stated before, RMA and similar methods can therefore be used.

The purpose is to declare an RTOS usable for hard real-time applications.

Performance measurements should encompass three different concepts: throughput, responsiveness, and determinism.

- The first one - throughput - is the speed at which a RTOS executes instructions. It gives a first perception of the suitability of the RTOS for a particular application. However, throughput is not sufficient to give adequate information on the performance of a system.

- For a more comprehensive view, the time needed for an OS to start handling an external event should be taken into account. This is the responsiveness of the system.
- Still, this is not enough. It remains to know how the system acts when it is heavily loaded, i.e. by how much the time needed to complete a particular operation varies when measuring the throughput and responsiveness under different workloads. The fact that the timings of the system's response are bounded is called determinism.

But more than just numeric results, the performance evaluation allows to understand the RTOS behavior in a more detailed way. From this study, conclusions can be made as to how certain features are implemented in the RTOS. It will answer other questions.

- How does the system handle tasks?
- Does this way of handling depend on the number of tasks in system; i.e. will it affect the system's performance?

These questions are applied to all the available system objects and operations.

Measurement procedure

The main problem is the use of a time reference for the measurements. Software timers often do not have enough precision. Their use adds large and unpredictable overhead to the measurements. Moreover, the measurements are synchronous with the system clock. Instead of a software timer, we use external hardware to carry out the measurements.

We decided to execute all tests on a "PC platform with a PCI bus interface" to be capable of using PCI stimu-

lator and bus analyzer equipment.

During the execution of a test, tracing data is written at a valid PCI bus address before and after the tested event. Writing the trace to a double word aligned address on a 33MHz PCI bus takes six to seven bus cycles, i.e. 180ns to 210ns. The trace is specific to its position in the code so the execution path can be followed when analyzing the results.

The PCI bus analyzer stores the data written on the PCI bus into local memory and adds a timestamp to it. When the test is completed, the data is downloaded from the PCI bus analyzer to a PC where it is further processed and analyzed.

As the collected data marks the start and the end of an event, it can be used to calculate the desired time intervals. Each test loops a minimum of 1000 times. This high number of iterations produces sufficient samples to analyze them using statistical tools. Figure 1 shows the generic performance measurement: the difference between traces T2 and T1 written on the PCI bus gives the wanted time interval.

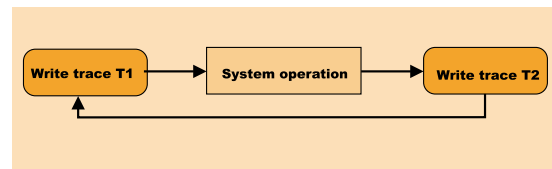


Figure 1. Generic performance measurement

For the interrupt testing series, we use the PCI bus stimulator. This board is able to generate interrupts on the PCI bus. The time interval between interrupts can be varied. Stress tests are also performed. Shortening the time interval between interrupts will finally crash the system.

Results layout

The results are presented in tables and graphs and are commented on. For each test series, the layout of the results is divided into five sections:

- A table containing number of samples, average, minimum, maximum, and standard deviation of the calculated time interval;
- A time graph, where the y-axis represents the measured time intervals and x-axis the timeline of the execution of the test. This graph gives an overview of the evolution and progress of the test. It is possi-

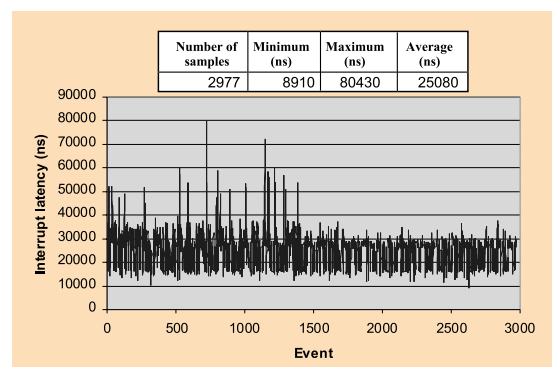


Figure 2. Interrupt latency for RTOS 1

ble to identify particular patterns, if there are any.

- A horizontal frequency distribution graph, which gives the distribution of the frequencies of the results.

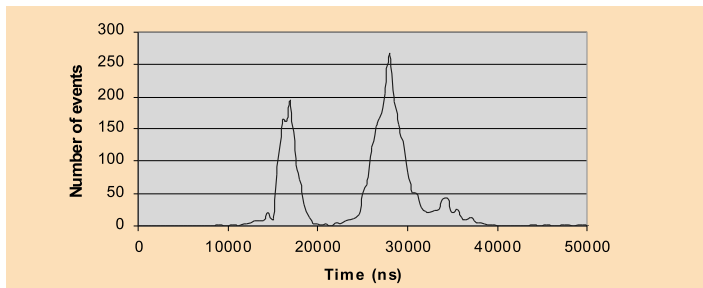


Figure 3 Interrupt latency frequency distribution for RTOS 1

- Detailed comments on the results;
- At the end of each test series, an additional section includes graphs comparing frequency values for different workloads.

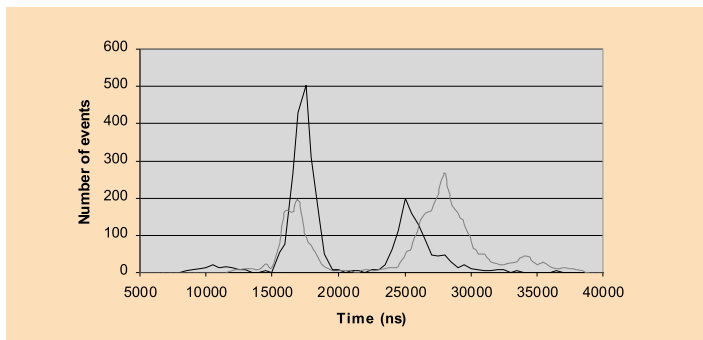


Figure 4. Interrupt latency frequency distributions for RTOS 1 under different workloads

Sometimes special paragraphs are inserted discussing network and disk performance under different load conditions. An example is given in Figure 5.

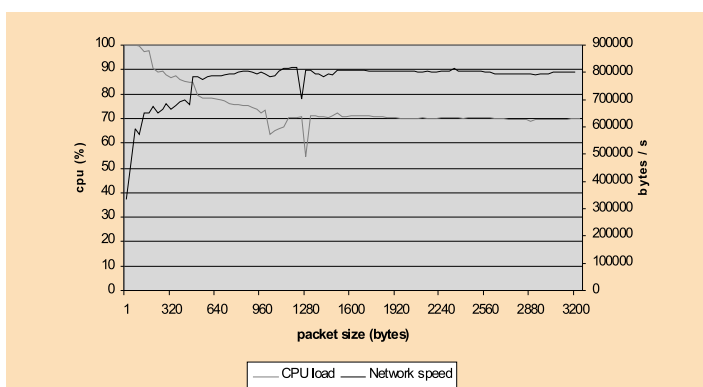


Figure 5 Network performance for RTOS 3

Several parameters (task switching latencies, acquiring semaphores, etc) will be tested for each RTOS with different system workloads.

System workload

Measurements have to be performed with different system workloads.

The system workload depends on one hand on the number of tasks in the system. For example, tasks requesting unavailable resources are queued by the system into system queues. The test series determine whether the time to reschedule tasks when releasing the requested

resource depends on the number of requesters or even more largely on the number of tasks ready to execute. Thus for the series on synchronization and communication objects each test measuring object requests is repeated with a different number of requesting tasks.

On the other hand, the system workload is influenced by the amount of data manipulated during context and task switches. To reduce the task switch latency, many RTOS implementations use light-weighted tasks called threads. Threads are light-weighted because they carry less information than processes. This implies that a thread control block is smaller than a process control block. The overhead caused by storing the control block of the pre-empted task and restoring the control block of the next running task is reduced. The amount of data contained in the task control block depends on the memory configuration. When tasks run in different memory space, the system needs to keep track of memory mappings for each task. This is not the case when tasks run in the same addressing space.

Memory configuration

Memory configurations can be classified into four models each being an enhancement of the precedent:

- No protection model: flat memory layout without memory protection between the tasks in the system.
- System/User protection model: the system address space is protected from the user address space. User processes and system processes run in a common virtual address space.
- User/user protection model: adds protection between user processes to the system/user model. User processes and the system process run in a common virtual address space.
- User/user private model: every user process is assigned its own private virtual memory.

The memory caches are disabled during a first execution round of the tests and enabled during a second round.

THE RTOS IN AN APPLICATION

Dedicated systems range from small devices like modems, washing machines, access systems to very large complex installations like fly by wire systems, air traffic control systems, switching systems and so on.

Despite the effort from all the vendors to make "scalable" products including the marketing efforts to sell an RTOS for whatever application, some RTOSs are not suited at all for some type of applications.

The knowledge and data gathered during the evaluation of the different products can easily be used to find the solution that is most qualified to accommodate the application's requirements.

CONCLUSION - THE PRODUCT EVALUATION

We have observed that the evaluations are really need-

ed. Indeed, the only feedback on the products today comes from customers using the product in just one particular application environment. No general conclusions can be drawn from this. Vendors distribute some benchmarks coming from different sources but no comparisons are possible due to the lack of standardized tests. Having some metrics is not enough.

Finally we hope that the availability of the product evaluations will help both vendors and users in making and defining better building blocks for the sake of enhancing the quality of the end products. ■

REFERENCES

- [1] J. Zalewski, "What Every Engineer Needs To Know About Rate-Monotonic Scheduling: A Tutorial," in Real-Time Magazine, Issue 1995/1 (<http://www.real-time-info.be>)
- [2] P. KORTMANN and G. MENDAL, PERTS Prototyping Environment for Real-Time Systems, Real-Time Magazine, Issue 1996/1 (ISSN 1018-0303).

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 Instituto 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.

* 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.