A Real Time Operating Systems (RTOS) Comparison

Rafael V. Aroca[1], **Glauco Caurin**[1]

[1]Laboratório de Mecatrônica
Escola de Engenharia de São Carlos (EESC) – Universidade de São Paulo (USP)
Av. Trabalhador São-Carlense, 400 – CEP 13566-590 – Caixa Postal 359
São Carlos – SP – Brasil

rafaelaroca@ieee.org, gcaurin@sc.usp.br

Abstract. *This article presents quantitative and qualitative results obtained from the analysis of real time operating systems (RTOS). The studied systems were Windows CE, QNX Neutrino, VxWorks, Linux and RTAI-Linux, which are largely used in industrial and academic environments. Windows XP was also analysed, as a reference for conventional non-real-time operating system, since such systems are also commonly and inadvertently used for instrumentation and control purposes. The evaluations include worst case response times for latency, latency jitter and response time.*

Resumo. *Este artigo demonstra resultados de uma análise quantitativa e qualitativa realizada em sistemas operacionais de tempo real (RTOS). Os sistemas estudados foram Windows CE, QNX Neutrino, VxWorks, Linux e RTAI-Linux, que são bastante utilizados em ambientes acadêmicos e industriais. O Windows XP também foi analisado, como uma referência de sistemas operacionais convencionais que não são de tempo real, já que estes tipos de sistema são usados comum e inadvertidamente para propósitos de instrumentação e controle. As avaliações incluem, mas não estão restritas aos piores casos de resposta de latência, jitter de latência e tempo de resposta.*

1. Introduction

Real Time Operating Systems (RTOS) are specially designed to meet rigorous time constraints. In several situations RTOS are present in embedded systems, and most of the time they are not noticed by the users.

A good example of this situation may be observed in the automobile industry, where it is estimated that 33% of the semiconductors used in a car are microcontrollers, being common for a car to have dozens of microcontrollers [Sangiovanni-Vincentelli et al. 2007, Parab et al. 2007]. As a consequence, the manufacturing costs of vehicles are reaching proportions that existed only in the aerospace industry, where 1/3 of the total cost of a vehicle is spent in the chassis, 1/3 in the powertrain and 1/3 in electronics [Wolf 2007].

Seeking for improvements on its products and development time reduction, the car manufacturers have been adopting RTOS to control the software that runs in the vehicles. A good example, is the electronic injection of fuel into the car engine, which must be made with rigorous time constraints. At each motor cycle, sensors need to measure and analyse the output gases generated by the combustion, and then compute the next mixture combination before the next ignition happens. Additionally, it is known that nowadays even simple motors, such as the ones used in motorbikes, already uses RTOS in their software.

Further examples are the recently developed avionics control systems, which use a single computer to cope with several aircraft subsystems, thus requiring an operating system with temporal and spatial partitioning systems [Krodel and Romanski 2007]. Spatial partitioning refers to tasks isolation in the computer memory, while temporal partitioning refers to tasks scheduling, dividing the processor time properly. These partitions allow a single processor to execute several tasks simultaneously, without the risk of one task causing interference in the temporal requirements of other tasks. This approach allows reduction of computers required to fly a plane, making it lighter.

In the international market, there are more than a hundred options of RTOS to choose from, while additionally there are free similar options as the Linux operating system. In this way, the decision on which system to use may be a key factor to the success or failure of a project, and the analysis must be made with impartiality and adequate criteria.

This work presents a brief comparison of several commercial and free RTOS through a qualitative and quantitative analysis. The experiments presented in this text were accomplished during a masters thesis research work, and are discussed here in a summarized form. More information and a detailed discussion of the tests can be found in [Aroca 2008].

The text is organized as follows: first, methodology is presented, next a brief qualitative description of each analyzed RTOS is made. Finally, a summary of the quantitative data presents the results of the fulfilled experimental study.

2. Methodology

According to Taurion, comparing RTOS is not a trivial task [Taurion 2005]. Besides this, the specialists from *Dedicated Systems*, an institution that has several projects and publications related to RTOS comparison, states that it is not possible to measure characteristics of a RTOS with reliability without using external hardware [Beneden 2001].

The most important factors of real time systems are the worst case response time of a task and worst case response time of an interrupt [Sohal 2001]. However, it makes no sense to analyze real time operating systems metrics such as interrupt latencies and task switching time without considering different CPU usage scenarios [Timmerman et al. 1998], as it is easier for a system to be more predictable when it is not overloaded.

Labrosse in [Labrosse 2002] states that the most important specification of a real time system is the amount of time that interrupts are disabled, because interrupt latency is a component of the system response time [Laplante 2004]. Additionally, response time measures of external interrupts gives a good idea of the real time capabilities related to a specific system or application [Franke 2007].

An evaluation approach proposed in several publications [Franke 2007, Barabanov 1997, Ganssle 2004, Köker 2007, Barbalace et al. 2008] consists in using the PC parallel port to receive an interrupt and generate a response to this interrupt, allowing testing the system as a black box. Using an external signal generator and an oscilloscope, it is possible to obtain the latency to handle interrupts and jitter (a random variation from one latency measurement to another), as one of the most accurate methods to measure execution time is through output ports [Stewart 2001]. Proctor also claims that latency tests only can be conducted by external means [Proctor 2001]. Taurion [Taurion 2005], also states that the most common operating systems metrics to measure quality is the task switching time between two processes and the latency until the start of an interrupt handler routine.

Keeping these facts in mind, it was decided to make the comparisons taking each tested system as a black box. The tests were conducted generating external stimuli with a signal generator, and analyzing the response for these stimuli with an oscilloscope. To guarantee the results reliability, all the experiments were executed in the same platform (a Pentium II 400MHz PC with 256MBytes of RAM memory) submitted to several different load scenarios (normal and overloaded use). Figure 1 depicts the configuration where the tests were made.

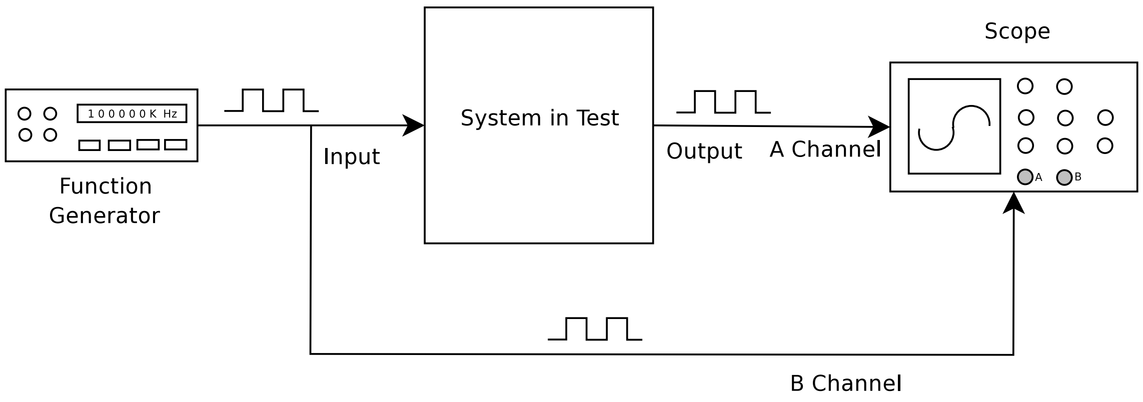

Figure 1. Experimental setup adopted for the tests

Although interrupt latency measurement through external mechanisms is commonly used, the International Society for Measurement and Control - ISA, through its committee for automation and control (Automation & Control Systems Committee - A&CS) do not recommend using latency as a measure of real time response [Dupré and Baracos 2001]. The test proposed by the A&CS consists in setting a system that copies the input signal directly to an output port, and measuring how many pulses were generated in the input and how many were copied to the output. Theoretically, while the system is stable, the accumulated number of pulses in both ports should be equal.

Later, the input signal frequency should be slowly incremented until the input and output pulses count starts to diverge. In this moment, the frequency should be reduced until the maximum system operation frequency is found. This frequency is the one that does not cause difference in the pulse count from the input to the output port. Still according to Dupré and Baracos, the obtained frequency is the inverse of the worst case response time. As a consequence of the considerations presented previously, the selected quantitative parameters to be analyzed in each system are:

1. **Latency:** Latency is analyzed externally taking the RTOS under test in conjunction with the hardware as a black box. The latency consists of the time difference

between the moment that an interrupt is generated and the moment that the associated interrupt handler generates an external response. The latency was measured in a scenario with low CPU use and with the CPU overloaded. For each scenario 60 independent samples were taken;

2. **Jitter:** Jitter is an indirect information obtained from several latency measures, consisting of a random variation between each latency value. In a RTOS, the jitter impact could be notorious, as it is analyzed by Proctor when trying to control step motors. For example, the pulses duration controls the motor rotation, but the jitter induce the torque to vary, causing step losses in the motor [Proctor and Shackleford 2001]. To compute jitter, the time difference between two consecutive interrupt latency measures is calculated. Finally, the greatest encountered difference is selected as the worst jitter of this system;
3. **Worst Case Response Time:** Worst Case Response Time is obtained using the method proposed by ISA that was discussed above analyzing the maximum interrupts frequency that is handled by the RTOS with reliability. The worst case response time is the inverse of the maximum frequency obtained. The test was made in a low CPU usage scenario and in an overloaded CPU scenario. For each scenario, 60 independent samples were taken.

Using the above mentioned technics, the obtained data were summarized and are presented in section 4 - Quantitative Summary.

3. Analyzed Systems

3.1. Microsoft Windows XP

Although Windows XP is not a RTOS, it is common to find several situations where this system is used to control critical applications [Stiennon 2008]. For this reason, Windows XP was included in this study. As Windows XP operating system may be influenced by software and different drivers, the caution of installing a new system was taken before running the tests. Windows XP can be installed only in x86 architecture computers (PC). The evaluated version had Service Pack 2.

Its task manager allows users or programmers to determine the priority of a running process. Between the options, there is one with the title “Real Time”. It is important to consider that this option does not offer real time capacity to a task. In fact, the choice of this priority level only configures the task scheduler to give the highest priority in the system to the task.

The system showed several instability situations when it was overloaded with hundreds of running tasks, a *ping flood*[1] against it and a high frequency of interrupts. This caused the whole system to crash showing a blue screen with the message “A problem has been detected, and Windows has been shut down to prevent damage to your computer”.

It is important to mention that the blue screen ocurrence was constant and the procedure which causes this error is well known: generating interrupts at 25KHz (or more) in the parallel port interrupt pin while a *ping flood* is executed against Windows XP.

[1]http://en.wikipedia.org/wiki/Ping_flood

In the experiments, Windows XP was stressed with *ping floods* that indirectly generates thousands of interrupts per second via network interface card joint with a fixed interrupt frequency imposed through the parallel port. The system showed stability and good real time response time up to a certain limit. Configuring the task priority to the ”real time” option of the scheduler was also very efficient, because when a real time task was consuming CPU time, all other tasks stopped responding. Even the mouse and keyboard did not answer movements or key hits, but the real time task performance did not deteriorate.

The conclusion is that given its restrictions and applications, Windows XP can be used with determinism and reliability in a real time system. This is confirmed by Cinkelj, who claims that it is possible to achieve data acquisition with soft real time guarantees in Windows XP when the computer is not overloaded [Cinkelj et al. 2005].

3.2. Microsoft Windows CE Embedded 6.0

The studied version of Windows CE is the Embedded 6.0. It supports ARM, MIPS, SH4 and x86 architectures. The tests were performed in the x86 architecture.

As in other operating systems, the clock interrupt is the “heartbeat” of the system [Viswanathan 2006]. In most systems, this is a constant rate interrupt generated by a hardware clock to trigger system’s house keeping routines, however Windows CE introduces an interesting innovation in this aspect: the variable clock tick, to reduce the overhead that the clock tick could cause in the operating system. For example, the variable clock tick system verifies that in a certain moment it is not necessary to generate clock tick interrupts at each 1ms, but only at each 100ms, changing the clock tick interrupt frequency. This allows the system to adjust the tick rate according to each situation [Viswanathan 2006]. This also implies in energy saving and more computing power.

One interesting Windows CE characteristic, is that Microsoft has made its source code available, giving developers more control and flexibility to the developed system. Another positive is that Microsoft has made available the complete development platform for this system without costs during four months from the moment the developer installs it, so that programmers can explore and test the system before really buying it.

Windows CE development platform also has several practical and powerful tools to verify if the system is meeting real time requirements and better debug the system. Among them, *Kernel Tracker*, *ILTiming* and *OSBench* are really useful.

The RTOS showed good stability in the frequencies measurement, even for the worst conditions. In addition, the input frequency was slowly improved until the maximum output value of the external generator (1MHz) was obtained, without causing any damage or problem in the running system. Meanwhile, the maximum input frequency that the system measured correctly was 50KHz. From this value, it is possible to compute the worst case response time of Windows CE which is 20μs (1/50KHz).

An interesting consideration about Windows CE is related to the overload test: to try to overload the system, dozens of programs were simultaneously executed, but there was a maximum limit reported by the system. In this situation, the operating system Graphical User Interface (GUI) showed a message informing that the system did not have more resources to start a new program, giving the option of selecting a program to be closed, to free resources to run the requested one. This feature is interesting, because it allows the system to preserve itself from unexpected states when the resources usage is too dangerous.

Even with all the *ping floods* and overloads, there was no situation that the system crashed requiring a reboot. In the worst overload scenario, the system stopped answering requests for some seconds, but right after it went back to normal operation.

The conclusion is that Windows CE Embedded 6.0 is a very robust and reliable operating system to execute real time tasks, with the advantage of offering several powerful development tools.

3.3. QNX Neutrino

QNX Neutrino is one of the most traditional RTOS in the market. Based in the microkernel architecture, it is completely compatible with the POSIX standards and has been certified by FAA DO-278 and MIL-STD-1553 standards. In the microkernel architecture, the kernel implements only four basic services: task scheduling, inter task communication, low level network communication and interrupt handling. All the remaining parts of the system (device drivers included) are implemented as user tasks, making the kernel fast, reliable and small. The tested version of Neutrino is 6.1.

One of the advantages of the microkernel over other types of kernels is that even when a critical error happens, such as in the filesystem for example, all the remaining parts of the system are not influenced. This means that the microkernel architecture offers a more robust environment than the ones used in other operating systems, although its problem is the overhead caused by the memory protection [Timmerman 2001] which have to be used very frequently, as all the parts of the system are isolated.

Neutrino supports ARM, MIPS, PowerPC, SH4 and the PC architecture. A recent feature of this operating system is called adaptative partitioning, which enables the creation of processor constraints by tasks. One example, is limiting the processor use of task A to no more than 30% of CPU time, and task B to no more than 10% of CPU time.

A study made by Dedicated Systems, compared QNX 6.1, VxWorks AE 1.1 (now discontinued) and Windows CE .NET (successor of Windows CE 3.0) concluding that all three systems are appropriate regarding their temporal behavior, however QNX 6.1 is superior to the competitor systems [Dedicated Systems 2002].

The executed tests showed predictability and reliability in this system. Even in the worst overload scenario, with hundreds of processes running at the same time and *ping floods* the system did not crash or stopped responding to requests. The input frequencies were increased to the maximum output value of the signal generator (1MHz), and in no situation the system showed deterioration in its determinism.

3.4. Micrium μC/OS-II

μC/OS-II stands for Microcontroller Operating System Version II consisting of a highly simplified, yet powerful real time kernel developed by Jean Labrosse, an embedded systems designer with more than 20 years of experience in the area. Labrosse is one of the most known authorities in the RTOS subject [Ganssle 2004]. According to his book that describes in details how μC/OS-II works, he argues that the decision of writing a new real time kernel from the beginning occurred because other commercial solutions were not meeting the quality requirements for the products he was developing [Labrosse 2002]. Today he owns a company called Micrium, which supplies this kernel. The tested version is the one that is included in the above mentioned book.

Ported to more than a hundred architectures [Ganssle 2004], including x86, μC/OS-II is mainly used in microcontrollers with low resources. This system was also certified by rigorous standards, such as RTCA DO-178B, which refers to the use of critical systems in aviation [Ganssle 2004, Labrosse 2002].

The real time kernel of μC/OS-II offers a reentrancy control mechanism and the priority inheritance protocol to avoid priority inversion, a common problem in real time kernels.

The tests showed that μC/OS-II has a good task scheduler, as the test to measure the external input frequency showed great results when the task had the greatest priority in the system (5), while its results became bad when the task was configured with the lowest priority available in the system (62). Even though, it was possible to measure input frequencies up to 520KHz, while the CPU load was near 99%. Higher frequencies did crash the system, requiring the computer to be restarted. This could be related to some interrupt counter overflow, or memory corruption, because the tested μC/OS-II kernel does not use the Memory Management Unit (MMU) to protect the tasks memory access from each other. A solution to this problem could be simply accomplished by using the new Micrium Real Time kernel released in 2008 called μC/OS-MMU, which uses the MMU to protect the tasks among each other.

The system exhibited little change in the measures when comparing overload scenarios to normal ones, with very low times. A final remark should be made about this system: the discussed kernel is not an operating systems with lots of services to the programmers - it is just a real time kernel, leaving most of the work to the programmer, who has to use external libraries in many situations, or buy additional packages from Micrium, such as TCP/IP stack, shell, or GUI.

3.5. Linux

Linux is a free operating system, with a modular monolithic kernel where all the important parts of the operating systems are in kernel space, such as memory management, task scheduler, filesystem and device drivers. It is possible to dynamically add or remove parts and functions of the kernel using Linux Kernel Modules (LKMs). Linux kernel implements memory protection with the MMU aid. The evaluated Linux kernel was 2.6.18.

Regarding real time systems, Linux is not a real time operating system, although, there is a low latency kernel patch called low-preempt patch that can be applied to the mainstream Linux to add soft real time capacity to the system. Linus Torvalds himself (Linux's author) affirmed in 2006 that he would use Linux to control a laser cutting machine using this patch.

However, adding more rigorous real time constraints, is not an easy task. Including hard real time guarantees in a kernel with millions of lines of code is very complex and could lead to errors. As the low-preempt patch is not fully adequate to transform Linux in a full real time kernel, better approaches should be used to solve this problem as it is discussed later in this article. Additionally, Ambike measured the clock resolution of popular systems such as Windows 2000 and Red Hat Linux 7.3, and obtained conclusive information to state that these systems are not good options for real time applications [Ambike et al. 2005].

Despite the fact that Linux is not a RTOS, it showed good temporal behavior, but when high frequencies were applied to the interrupt input pin joint with *ping flood*, the system became unstable and crashed. The jitter was also relatively high, and could cause unexpected variations in real time systems that needs precision.

3.6. Real Time Application Interface (RTAI)

Real Time Linux has been recently offering a good cost/benefit relation in the real time area, being used in small applications, tests, and even in medical and state of the art scientific equipments [Irwin et al. 2002, Katuin 2003]. According to Weinberg, Real Time Linux is good enough to meet the requirements of 95% of real time applications [Weinberg 2001]. The evaluated RTAI version was the 3.4.

Real Time Linux is so well consolidated that well known institutions with RTOS needs have been using Real Time Linux in some applications. Examples are the American National Institute for Standards and Time (NIST) [NIST 2002] and the National Aeronautics and Space Administration (NASA) [Kalynnda 2002]. NASA itself, already used Real Time Linux in production to build a radar that collects data from tornados [Yodaiken 1999].

Although there are several different versions of Real Time Linux, the chosen version for analysis is RTAI. According to Kim and Ambike, RTAI has real time performance comparable to the best RTOS such as VxWorks and QNX [Kim et al. 2006], having sufficient determinism to replace them [Barbalace et al. 2008], however it does not offer any certification or guarantee to its users, as it is a free software.

RTAI approach consists in isolating the operating system into domains, making the real time domain as the one with highest priority. Whenever a real time task needs to be executed, a tiny scheduler, also called a nanokernel, schedules this task, freezing the whole remaining system. When there are no pending real time jobs to be done, the other parts of the system, such as GUI and user interface are executed [Barabanov 1997]. This solution is excellent, as most real time systems consists of a combination of tasks with soft and hard real time requirements [Labrosse 2002]. This enables the use of a single computer to control both real time critical functions and user interface, networking or others features that can be added at any time without changing the real time performance.

The tests showed that the system division into domains gave a good stability and determinism to the environment, in a way that the real time tasks had high priority and reliability.

3.7. Wind River VxWorks

VxWorks is the most used RTOS in the embedded systems industry [Ip 2001, Cedeno and Laplante 2007, Timmerman 2000b, Timmerman 2000a]. Examples of applications where VxWorks is used is the international space station [Cedeno and Laplante 2007], and in the famous NASA rover robots *Spirit* and *Opportunity*. This RTOS had been certified by several agencies and international standards for real time systems, reliability and security-critical applications. The evaluated version was VxWorks 6.2 which takes advantage of the system's MMU to isolate and protect the tasks.

A curious fact happened to the Mars rovers and helped show how flexible VxWorks is. When the robots arrived in the Martian ground, a software bug happened and the system software became unstable causing a watchdog to reset the robot's computer constantly. Using a remote debugging system, the developers were able to diagnose and fix the problem 70 million kilometers away from the robot. The problem consisted of a priority inversion and was easily solved switching on the VxWorks priority inheritance protocol.

VxWorks kernel allows tasks to be grouped by functionality and contained in Run Time Processes (RTP). Each RTP is structured to isolate tasks from each other in order to protect the system in the case one of them fails. Besides VxWorks 6.x, several other products are based upon VxWorks: VxWorks 653 is designed to follow complete ARINC 653[2] conformance for aerospace and defense industries with safety and security requirements for mission-critical applications. VxWorks MILS [3] is also intended for Aerospace and Defense, adding high assurance for secure information sharing. VxWorks MILS allows a system to run applications at different security levels, providing assurance that security levels and communication are shared, or not shared according to strict policies.

VxWorks 6.2 was overloaded with 400 tasks while a ping flood was executed against the tested machine. In this scenario the system could handle and measure with reliability a 260KHz input frequency.

With a worst response time of 3,85μs, the analysis concluded that VxWorks 6.2 RTOS is between the ones with the smallest responses time. Its small jitter also shows that this system is very predictable.

4. Quantitative Summary

Table 1 presents the experimental results for each system considering the worst measured value for each of them with the system overloaded during the measurements. Line A consists of worst response time (1/max frequency of stable operation), line B of interrupt latency and line C consists of the latency jitter.

One thing that was notable is the low response time of μC/OS-II. It should be noted that the tested version of the system did not use the system's Memory Management Unit (MMU). This improves the system performance, nevertheless there is no protection between the memory areas of the tasks, improving the possibility of a task to corrupt others, or even the whole system.

The values showed in the table can be compared with a criteria that defines a hard real time system according to OMAC (Open Modular Architecture for Control[4]) user group that considers a system "hard real time" the one that has a jitter no higher than $100\mu s$ in tasks that has cycles of up to 10ms [Hatch 2006].

[2]Avionics Application Standard Software Interface
[3]MILS (Multiple Levels of Security)
[4]More information at http://www.omac.org/

In this aspect, all the systems, with the exception of Windows XP, fit in the definition of a hard real time RTOS according to OMAC.

Table 1. Worst times measured during the experiments. A: Response Time (1/maximum sustained frequency), B: Latency, C: Latency Jitter

	Win XP	Win CE	Neutrino	μC/OS-II	Linux	RTAI	VxWorks
A	200μs	20μs	20μs	1,92μs	13,89μs	5μs	3,85μs
B	848μs	99μs	35,2μs	3,2μs	98μs	11,4μs	13,4μs
C	700μs	88,8μs	32μs	2,32μs	77,6μs	7.01μs	10,4μs

5. Conclusion

In this work, real time operating systems were compared through several parameters, and it was noticed that with the exception of Windows XP, which is not a RTOS, all the studied systems have met the temporal requirements in a satisfactory way.

The well consolidated systems QNX Neutrino and VxWorks from Wind River, did really show determinism and reliability, although two newer systems that are not widespread also showed promising characteristics: Windows CE and RTAI Linux.

Windows CE Embedded was written from scratch especially for critical applications, and during the tests it behaved as a robust, powerful and flexible system. In the free open source domain, RTAI offers the opportunity of implementing reliable real time systems with free software, having all the advantages of the Linux community and already available free software that could be used together.

A final consideration of this work is that there is a very rich field involving the choice of the most suitable RTOS for mission critical or noncritical embedded tasks. In this work, just some technical aspects were taken into account, but it is well known that subjective aspects also plays an important role in the choice of a RTOS. How familiarized is the development team to a specific RTOS? How experienced, and finally how big is their resistance to change to a different RTOS?

6. Acknowledgments

The authors would like to thank Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) for its support and the companies that provided the RTOS for tests and evaluation (Wind River Systems, Microsoft, Micrium and QNX Software Systems).

References

Ambike, A., Kim, W.-J., and Ji, K. (8-10 June 2005). ”real-time operating environment for networked control systems”. *American Control Conference, 2005. Proceedings of the 2005*, pages 2353–2358 vol. 4.

Aroca, R. V. (2008). Análise de sistemas operacionais de tempo real para aplicações de robótica e automação. Master’s thesis, Escola de Engenharia de São Carlos (EESC) - Universidade de São Paulo (USP).

Barabanov, M. (1997). A linux-based realtime operating system. Master’s thesis, New Mexico Institute of Mining and Technology.

Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., and Taliercio, C. (2008). "performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application". *Nuclear Science, IEEE Transactions on*, 55(1):435–439.

Beneden, B. V. (2001). "windows ce 3.0: Breathing down rtos vendors' necks". *Dedicated Systems Magazine*.

Cedeno, W. and Laplante, P. A. (2007). "an overview of real-time operating systems". *Journal of the Association for Laboratory Automation*, 12:40–45.

Cinkelj, J., Mihelj, M., and Munih, M. (20 May 2005). "soft real-time acquisition in windows xp". *Intelligent Solutions in Embedded Systems, 2005. Third International Workshop on*, pages 110–116.

Dedicated Systems (2002). Comparison between qnx rtos v6.1, vxworks ae 1.1 and windows ce .net. Technical report, Dedicated Systems.

Dupré, J. K. and Baracos, P. (2001). Benchmarking real-time determinism. Technical report, Instrumentation, Systems &. Automation Society (ISA).

Franke, M. (2007). A quantitative comparison of realtime linux solutions. Technical report, Chemnitz University of Technology.

Ganssle, J., editor (2004). *The Firmware Handbook*. Elsevier.

Hatch, J. (2006). Windows ce real-time performance architecture. In *Windows Hardware Engineering Conference*.

Ip, B. (2001). Performance analysis of vxworks and rtlinux. Technical report, Columbia University, Department of Computer Science, New York. http://www1.cs.columbia.edu/ sedwards/classes/2001/w4995-02/reports/ip.pdf.

Irwin, P., Richard, L., and Johnson, J. (2002). Real-time control using open source rtos. In Lewis, H., editor, *Advanced Telescope and Instrumentation Control Software II*, volume 4848, pages 560–567. SPIE.

Kalynnda, B. (2002). Real-time linux evaluation. Technical report, Glen Research center, NASA.

Katuin, J. (12-16 May 2003). "proton therapy treatment room controls using a linux control system". *Particle Accelerator Conference, 2003. PAC 2003. Proceedings of the*, 2:1068–1070 Vol.2.

Kim, W.-J., Ji, K., and Ambike, A. (2006). Real-time operating environmentfor networked control systems. *Automation Science and Engineering, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, 3(3):287–296.

Krodel, J. and Romanski, G. (2007). Real-time operating systems and component integration considerations in integrated modular avionics systems report. Technical report, U.S. Department of Transportation - Federal Aviation Administration.

Köker, K. (2007). *Autonomous Robots and Agents*, chapter Embedded RTOS: Performance Analysis With High Precision Counters, pages 171–179. Springer Berlin / Heidelberg.

Labrosse, J. (2002). *MicroC/OS-II - The Real Time Kernel*. CMP Books, 2 edition.

Laplante, P. A. (2004). *Real-Time System Design and Analysis*. John Wiley & Sons.

NIST (2002). Introduction to linux for real-time control. Technical report, National Institude of Standards and Technology (NIST).

Parab, J. S., Shelake, V. G., Kamat, R. K., and Naik, G. M. (2007). *Exploring C for microcontrollers - A Hands On Approach*. Spring.

Proctor, F. M. (2001). Measuring performance in real-time linux. In *Third Real-Time Linux Workshop*.

Proctor, F. M. and Shackleford, W. P. (2001). Real-time operating system timing jitter and its impact on motor control. In *Proceedings of the SPIE Conference on Sensors and Controls for Intelligent Manufacturing II*.

Sangiovanni-Vincentelli, A., Sangiovanni-Vincentelli, A., and Di Natale, M. (2007). ”embedded system design for automotive applications”. *Computer*, 40(10):42–51.

Sohal, V. (2001). How to really measure real-time. In *Embedded System Conference*.

Stewart, D. (2001). Measuring execution time and real-time performance. In *Embedded System Conference*.

Stiennon, R. (2008). Top ten worst uses for windows. On Line. Available at: http://www.networkworld.com/community/node/29644?ts. Last Access: August/2008.

Taurion, C. (2005). *Software Embarcado - A nova onda da informação*. Brasport.

Timmerman, M. (2000a). ”rtos market overview - a follow up”. *Dedicated Systems Magazine*.

Timmerman, M. (2000b). ”rtos market survey - preliminary results”. *Dedicated Systems Magazine*.

Timmerman, M. (2001). What makes a good rtos. Technical report, Dedicated Systems.

Timmerman, M., Beneden, B. V., and Uhres, L. (1998). ”rtos evaluations kick off!”. *Real-Time Magazine*, 98-3:6–10.

Viswanathan, S. (2006). Understanding the windows ce variable tick timer. Technical report, Microsoft Corp.

Weinberg, B. (2001). ”embedded linux - ready for real time”. *Third Real-Time Linux Workshop*, (3).

Wolf, W. (2007). ”the embedded systems landscape”. *Computer*, 40(10):29–31.

Yodaiken, V. (1999). The RTLinux manifesto. In *Proc. of The 5th Linux Expo, Raleigh, NC*.