International Journal of
Science and Engineering Investigations vol. 6, issue 63, April 2017
ISSN: 2251-8843

Comparative Analysis of Real-Time Operating System (RTOS) of Some Selected OS Using External Signal Generator and Oscilloscope

A. Abdulganiyu[1], I. Rabiu[2]
[1,2]Department of Mathematics/Computer Science, Faculty of Natural Sciences, Ibrahim Badamasi Babangida University, Lapai
(abdulg2009@yahoo.com)

Abstract- This study presents a quantitative and qualitative comparative analysis of Real Time Operating systems (RTOS) of some selected operating systems in order to determine their performance in executing a task(s) over real time. In so doing, the studied systems which include Windows XP, Window 8, Window 7 professional and window 10 which are largely used in industrial and academic environments were selected and analysed using a function generator and Oscilloscope connected to the analysed system as a reference for conventional non-real-time operating system. The evaluations from the setup include real run time, worst case response times for latency, latency jitter and response time. Results from this study will be used as a generalization for the performance of such operating system on real time and thus, a consideration from this work will inform the choice of the most suitable RTOS for mission critical or non-critical embedded tasks.

Keywords- Real Time Operating Systems, Window, Oscilloscope, Function Generator

I. Introduction

Real time applications have become a very common phenomenon in the past few years for both software developers and the end users. Developers are given the enviable task of making software with real time constraints. For end users, a large number of real-time operating systems (RTOS) are available in the market and one does get confused as to which one to use such that it provides the best overall benefits in terms of cost and operability. There are set of certain benchmarks, which one could examine in a RTOS, such as latency, susceptibility to different loads. Real Time Operating Systems (RTOS) are specially designed to meet multitasking and rigorous time constraints. In several situations RTOS are present in embedded systems, and most of the time they are not noticed by the users. Real time operating systems are the multitasking operating systems, which not only depend upon the logical correctness but also depend upon the application delivery time. These valuable RTOS works on the philosophy of the round robin algorithm and preemptive priority scheduling method. The Idea behind the operating system is not very new, it's many years old. Evolution of operating system causes significant changes in task solving methodology. They stay's responsible for the overall system requirement, performance, and task solving methodology. A system which works on the aspect of time determination is generally known as the real time system. Advancement of embedded based real time operating system guarantees the time constraint capability and predictability of an application. Similarly, embedded systems are becoming an integral part of commercial products today. Mobile phones, watches, flight controllers etc. There is a strong and compatible relationship between the system hardware and the software, primarily the operating system to ensure hard real time deadlines. The real time operating system has to interface communicate well with the hardware below it to prevent casualty.

On the other hand, various OS vendors now employed various stringent standards that may not meet the needs required for high level applications(EMF, 2015), Thus, the question arises that, are this certified RTOS truly necessary for the high level applications? If yes, than which one is the suitable platform for a particular application? The objective of this study is to enlighten both professional group and especially non-technical group on best OS that may have the best RTOS by providing a comparison chart among various popular RTOS.

In this paper, the enhancement of operating system in real time environment will be discussed based on the experimental result analysis. It highlight the freely available, real-time operating systems echo's and analyse the real time attributes, like timing latency, context switch latency and interrupt latency, of these operating systems by means of simple applications. The approach will attempt to introduce the emerging trends in this field and provide a user friendly classification, which can cover more than one professional operating system. It will begin with the conceptual enhancement of the current technology at fundamental level for better technical understanding. The classification provides choices to system designer, student and researchers. Hence, this paper will present a brief comparison of several commercial and free RTOS through a qualitative and quantitative experimental analysis.

II. METHODOLOGICAL APPROACH

The methodological approach of this study is based on the evaluation approaches proposed in several publications by different authors which include Franke (2007 professional), Barabanov (1997 professional), Ganssle (2004), Koker (2007 professional), Barbalace et al. (2008) in this scheme, various operating systems such as XP Windows (Window 7 professional , Window 8, Window 10 and Window 10) will be tested for real time performance, thus, in doing so, the experimental set will employ a PC parallel port to receive an interrupt and generate a response to this interrupt, allowing testing the system as a black box. Therefore, using an external signal generator and an oscilloscope, the execution time for the various windows tested will measured and compared for analysis.

These tests is conducted such that the signal generator will generate an external stimuli thus, analyzing the response for these stimuli with an oscilloscope. To guarantee the results reliability, all the experiments will be executed in the same platform (a Pentium IV 400MHz PC with 256M Bytes of RAM memory) subjected to several different load scenarios (normal and overloaded use).

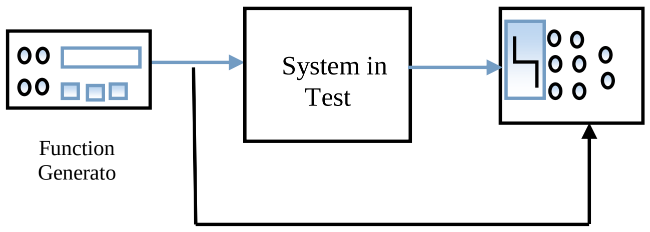

Figure 1. Schematic Model of the Experimental Setup for the proposed study

III. APPROACHES TO RTOS TEST

According to Taurion, comparing RTOS is not a trivial task. 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. 110108], 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 [Franke2007 professional].

An evaluation approach proposed in several publications [Franke2007 professional, Barabanov 1997 professional, Ganssle2004, K¨oker2007 professional, and Barbalaceetal. 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 [Stewart2001]. Proctor also claims that latency tests only can be conducted by external means [Proctor 2001]. 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 256M bytes of RAM memory) submitted to several different Load scenarios (normal and overloaded use).

IV. MATERIALS USED

In carrying out the experiment the following instrument and systems were used, this enables the execution of Real-time with different versions of Operating Systems.

V. MICROSOFT WINDOWS XP, 7 PROFESSIONAL, 8 AND 10

In subjecting the objectives of this study to text, various operating systems were used this include Window XP, 7 professional, 8, and 10. 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 these Window operating systems 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 Pack2.

VI. FUNCTION GENERATOR

A function generator is usually is piece of electronic test equipment or software used to generate different types of electrical waveforms over a wide range of frequencies. Some of the most common waveforms produced by the function generator are the sine, square, triangular and saw tooth shapes. These waveforms can be either repetitive or single-shot (which requires an internal or external trigger source). Integrated circuits used to generate waveforms may also be described as function generator ICs.

Although function generators cover both audio and RF frequencies, they are usually not suitable for applications that need low distortion or stable frequency signals. When those traits are required, other signal generators would be more appropriate.

VII. Oscilloscope with connecting cables

An oscilloscope, previously d informally known as a scope, CRO (cathode-ray oscilloscope), or DSO (for the more modern digital storage oscilloscope), is a type of electronic test instrument that allows observation of constantly varying signal voltages, usually a1so a two-dimensional plot of one or more signals as a function of time. Other signals (such as sound or vibration) can be converted to voltages and displayed.

Oscilloscope is used to observe the changes of an electrical signal over time, such that voltage and time describe a shape which is continuously graphed against a calibrated scale.

VIII. Results and Analysis

As a result of the various selections and considerations of the chosen parameters for the experiment presented in this research, 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 scenario60 independent samples were taken

2. Jitter: Jitter is 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 Shackle ford 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.

IX. Analyzed Systems

A. *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 Pack2. 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 flood1 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 occurrence was constant and the procedure which causes this error is well known: generating interrupts at 25 KHz (or more) in the parallel port interrupt pin while a ping flood is executed against Windows XP.

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].

B. *Window 7 professional .0 embedded*

The studied version of Window7 professional .0 is the embedded type. 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 housekeeping routines; however Windows 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 interrupt sat 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 7 professional 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 7 professional .0 development platform also has several practical and powerful tools to verify if the system is meeting real time requirement and better debug the system.

Among them, Kernel Tracker, IL Timing and OS Bench 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 was50KHz. From this value, it is possible to compute the worst case response time of Windows which is 200µs (1/50 KHz).

C. *Window 8.0*

The real time kernel of this window 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 the window 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, while its results became bad when the task was configured with the lowest priority available in the system). Even though, it was possible to measure input frequencies up to 520 KHz, while the CPU load was near 1010%. Higher frequencies did crash the system, requiring the computer to be restarted. This could be related to some interrupt counter over flow, or memory corruption, because the tested window 8.0 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 2012, 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.

D. *Window 10.0*

The Window 10.0 used in this research has is system memory re-configure in such that it becomes 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, file system and device drivers. It is possible to dynamically add or remove parts and functions of the kernel using Kernel Modules (KMs). Kernel implements memory protection with the MMU aid the evaluated kernel was 2.6.18.

Regarding real time systems, window 10.0 is not a real time operating system, although, there is a low latency kernel patch called low-pre-empt patch that can be applied to the main stream to add soft real time capacity to the system. 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-pre-empt patch is not fully adequate to transform window 10.0 in a full real time kernel, better approaches can be used to solve this problem. Additionally, Ambike measured the clock resolution of popular systems such as Windows 2000 and Red Hat Linux 7 professional .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 window 10.0 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 need precision.

X. Experimental Results

Table 4.1 shows the experimental results for each system considering the worst measured value for each of them with the system overloaded during the measurements. Line 1 consists of worst response time (maximum frequency of stable operation), line 2 of interrupt latency and line 3 consists of the latency jitter. One thing that was notable is the low response time of Window 10.0. 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) user group that considers a system "hard real time" the one that has a jitter no higher than100µs in tasks that has cycles of up to 10ms.

TABLE I. Summary of Time Measured from the experiment

S/No.	Tested Windows/Operating Systems			
	Win Xp	Win 7 professional .0	Win 8.0	Win 10.0
1	200μs	27 professional μs	3.56μs	110.88μs
2	7 professional 100 μs	110μs	6.7 professional µs	107 professional μs
3	650 μs	108.5μs	2.45μs	810.110μs

Where
1: Response Time (maximum sustained frequency),
2: Latency,
3: Latency Jitter

XI. Numerical Interpretation

The following charts are the numerical interpretation of the results obtained for each window

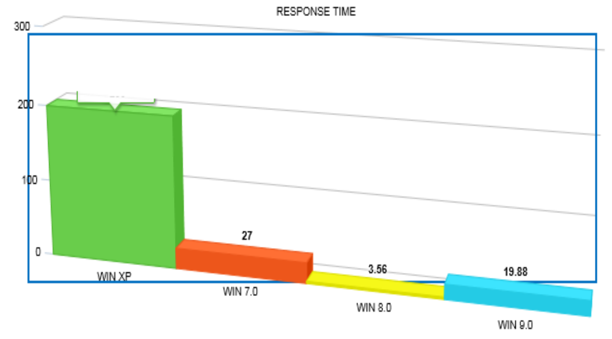

Figure 2. Response time for all the measured systems showing variations of worst case response time in Terms of task execution

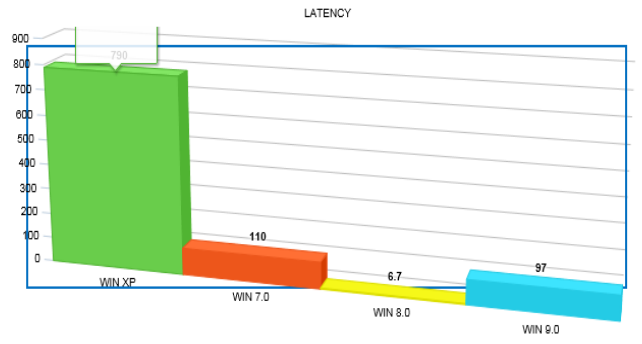

Figure 3. Latency for all the measured systems showing variations of time in Terms of task execution and over load

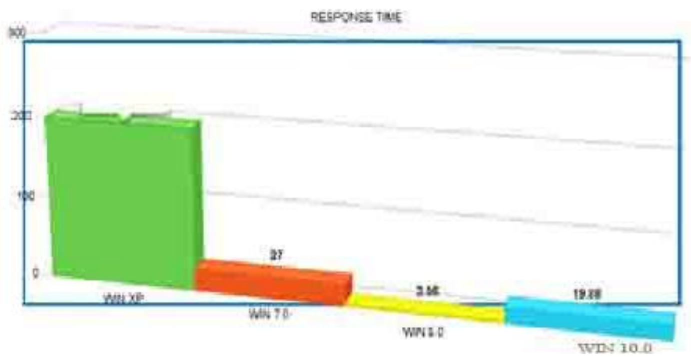

Figure 4. Latency Jitter for all the measured systems showing variations of time in terms of task execution and over load

XII. Window Values of Response time, Latency and Latency Jitter

The following figures shows the tested window values for response time, latency and latency jitter, thus the performance of each window were measured under different load conditions and under different tasks.

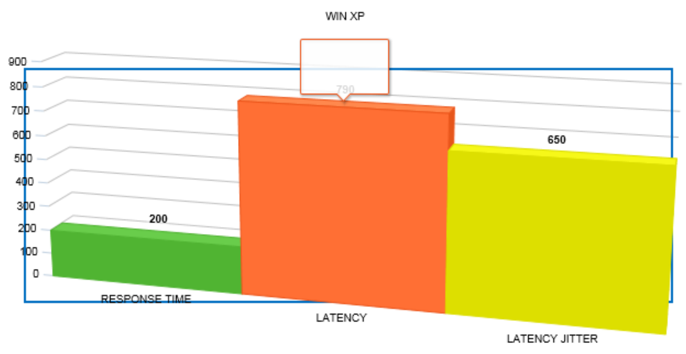

Figure 5. Latency, Latency Jitter and response time for window XP under different load conditions measured with time

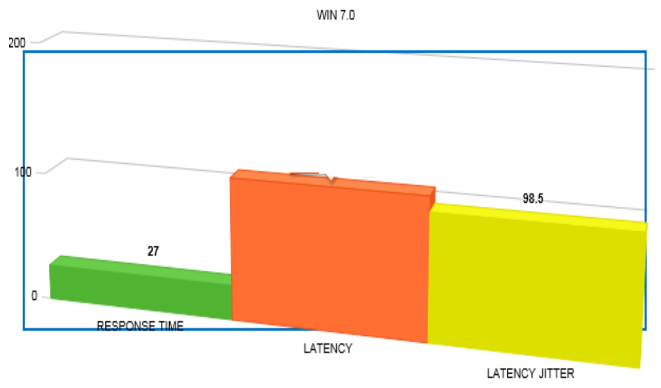

Figure 6. Latency, Latency Jitter and response time for window 7 professional .0 under different load conditions measured with time

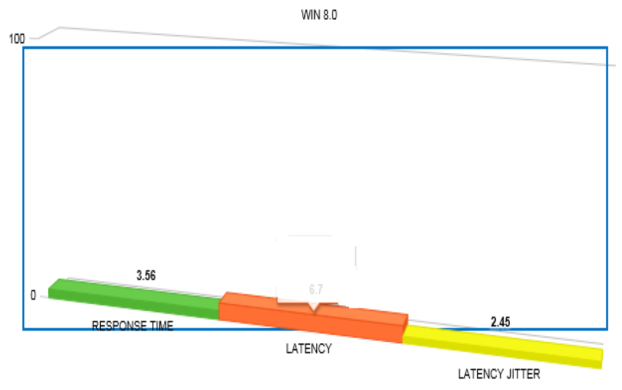

Figure 7. Latency, Latency Jitter and response time for window 8.0 under different load conditions measured with time

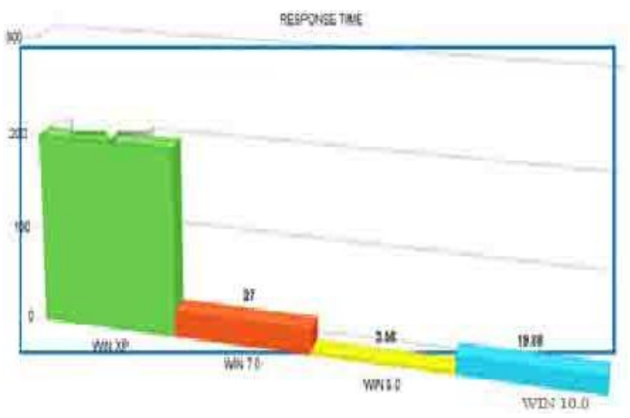

Figure 8. Latency, Latency Jitter and response time for window 10 under different load conditions measured with time

XIII. Discussion of Findings

The test on Windows XP operating system used in this study was influenced by software and different drivers. During this test the system showed several instability situations when it was overloaded with hundreds of running tasks and a high frequency of interrupts. This caused the whole system to crash showing a blue screen message. It is important to mention that the blue screen occurrence was constant and the procedure which causes this error is well known thus, the interrupts in the parallel port is executed against Windows XP.

Therefore, when a fixed interrupt frequency is imposed through the parallel port, the system showed stability and good real time response time up to a certain limit of 200μs, Latency of 7 professional 100μs and latency jitter of 650μs. In conclusion, it can be stated that given any restrictions and applications, Windows XP can be used with determinism and reliability in a real time system.

Similarly, the Window7 professional .0 embedded tests were performed in the same way with other operating systems, the clock interrupt was used as the “heartbeat” of the system. This allows the system to adjust the tick rate according to each situation. During the test for this system is was discovered that the system stopped answering requests for some seconds, but right after it went back to normal operation (worst overload scenario). The scenario gave rise to a response time of 27 professionalμs, latency of 110μs and latency jitter of 108.50μs. Therefore, in conclusion is that Windows 7 professional, 0 embedded is a very robust and reliable operating System to execute real time tasks, with the advantage of offering several powerful development tools.

Also, window 8.0 operating system exhibited little change in the measures when comparing overload scenarios to normal ones, with very low times. The response stood at3.56μs, latency at $6.7\ professional\ \mu s$ and latency jitter at 2.45μs

Finally, the test on Window 10.0 used in this research has is system memory re-configure in such that it becomes a free operating system, Despite the fact that the window 10.0 is not a RTOS, it showed good temporal behavior , but when high frequencies were applied to the interrupt input, the system became unstable and crashed showing blue screen. The jitter was also relatively high, and could cause unexpected variations in real time systems that need precision. The latency stood at 107 professionalμs, latency jitter at 810.110$\mu s$and response time at 110.88μs.

XIV. Conclusion

In this study a performance comparison had been made on four selected systems with different operating system (Window 7 professional, Window 8, and Window 10) by comparing their RTOS. In so doing, a function generator and an oscilloscope were used to measure the response time, latency as well as latency jitters of each operating system. The experimental results for each system were recorded by considering the worst measured value for each of them with the system overloaded during the measurements. The values obtained were tabulated and analyzed. The results from the various computation shows that even with same system specification the RTOS will varies with accordance to the type of operation system also the 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. Similarly, 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) user group that considers a system ”hard real time” the one that has a jitter no higher than100µs in tasks that has cycles of up to 10ms [Hatch 2006].

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 window 7 professional .0, window 8.0, window 10.0 did really show determinism and reliability, although at the beginning the windows shows instability but at the end they all showed promising characteristics.

Windows 7 professional Embedded was tested for critical applications, and during the tests it behaved as a robust, powerful and flexible system. In the free open source domain, RTAI of this wind could offers the opportunity of implementing reliable real time systems with software, having all the advantages of the Linux community and already available 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 a subjective aspect also plays an important role in the choice of a RTOS.

Conclusively, the following recommendations are proffered both at professional or technical levels:

1. Response is one integral part of any effective system particularly those uses for high level programming, therefore, latency and latency jitters should always be tested as this

allows a quick response time to a maximum frequency of stable operation.

2. One thing that was notable is the low response time of μC/OS for window 8. 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.

3. System performance under different load conditions should be considered before judging which RTOS better than the other RTOS. This will enhance and inform choice of users.

4. Since RTOS require different time to execute there should be different open room to think of different ways of optimizing a Kernel for Real time applications by taking the best features of each.

5. Since RTOS are evaluated for performance, them it should have some support for multitasking (threads) and it should be pre-emptive priority driven system.

RTOS should support thread synchronization using semaphores or mutexes. RTOS must have sufficient number of priority levels as such RTOS must avoid priority inversion.

REFERENCES

[1] Edwards. S (2001).Languages for Digital Embedded systems

[2] Franke, M. (2007): "A quantitative comparison of real-time Linux solutions". Technical report, Chemnitz University of Technology.

[3] Keeling N.J. (1998). How Priority Inversion messes up real-time performance and how the Priority Ceiling Protocol puts it right. Real-time Magazine 1010(4): 46-50.

[4] K"oker, K.(2007). "Autonomous Robots and Agents, chapter Embedded RTOS: Performance Analysis with High precision counters, pages 171-179, Springer Berlin /Heidelberg.

[5] Labrose, J. (2002). MicroC/OS-II-The Real Time Kernel, CMP Books, 2 edition.

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

[7] NIST (2002). Introduction to Linux for real- time control, Technical report, National Institude of standards and Technology (NIST).

[8] Liu C. and Layland J. (2002) Scheduling algorithm for multiprogramming in a hard real-time Environment, Journal of the Association for Computing Machinery, 20(1): 46-61

[9] Stewart D. (2001) Measuring Execution Time and Real-Time Performance, In Embedded Systems conference, San Francisco, April 2001.

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

[11] Obenland K. (2001) Real-Time Performance of Standards based Commercial Operating Systems. In Embedded Systems conference, San Francisco, April 2001.

[12] ManasSaksens (2001), Linux as Real-Time Operating System. In Embedded Systems conference, San Francisco, April 2001.

[13] Martin Timmerman (1998). RTOS Evaluations Kick Off. Real-Time Magazine 108(3): 6-10. March 1998.

[14] Taurion, C. (2005). Software Embarcado-A nova onda da informac, ao, Brasporat.

[15] Timmerman, M. (2000a). "RTOS market overview – a follow up", Dedicated Systems Magazine.

[16] Timmerman, M. (2000b), "RTOS market survey – preliminary results", Dedicated systems Magazine.

[17] Timmerman M. (2001), What makes a good RTOS. Technical report, Dedicated Systems.