1. Introduction

In this article we want to expose the results of a benchmark concerning real time performances of three RTOS: VRTXsa, RTAI and RTLinux.
The goal of the project was to evaluate the possibility to migrate the operating system of an embedded telecommunication SBC from VRTXsa to a linux based one.
This project has been realized as thesis work for a famous telecommunication company and has evolved under the supervision of the Rome's University "La Sapienza".

2. Tests

We executed hardware tests and software tests with different modalities to evaluate all parameters described above. This is the platform we used to execute these tests:
Motorola Freescale PQ2FADS-VR
CPU: PowerPc 8275 @ 200 Mhz - bus 66 Mhz.
RAM: 32 Mb
FLASH: 8 Mb.
Tests have been executed without load and with two different load solutions in order to evaluate differences between the executions in relationship with the changing load.

2.1 Hardware Tests

In hardware tests we use oscilloscope to measure hardware latency time: time which elapses between the generation of an interrupt and the moment its handler starts.

2.1.1 Test modality

Hardware test have been done using oscilloscope. We made a little hardware modification: we built a bridge between LD13 led and C11 pin (of system expansion connector). C11 pin is the interrupt request pin and is connected with irq 6 signal (DP6/CSE0/IRQ6#). Connection created between LD13 led and C11 pin is fundamental to reply infinitely the interrupt generation and its handling. Test program uses LD13 led pin to generate tension levels of +5V/0V. Installing a new Interrupt Service Routine on interrupt 6 is possible to handle this interrupt. Turning on LD13 led we'll get a transition High → Low of the tension level. Such event generate interrupt 6 and handler is started. The handler we defined will turn off, and on afterwards, LD13 led generating in this way a new interrupt and so on. In this way an infinite sequence of interrupt is generated. Applying the oscilloscope probe on the connection it's possible to analyse the wave form and to estimate with a good precision the time which elapses between the event and its handler starting. This time includes hardware latency and interrupt latency.

Wave form on the monitor will be similar to the following one:

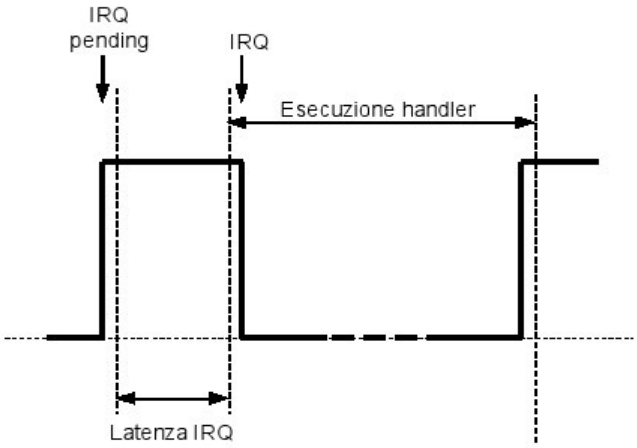


Figura 2: esempio di forma d'onda visibile sull'oscilloscopio.

2.1.2 RTAI Measurements

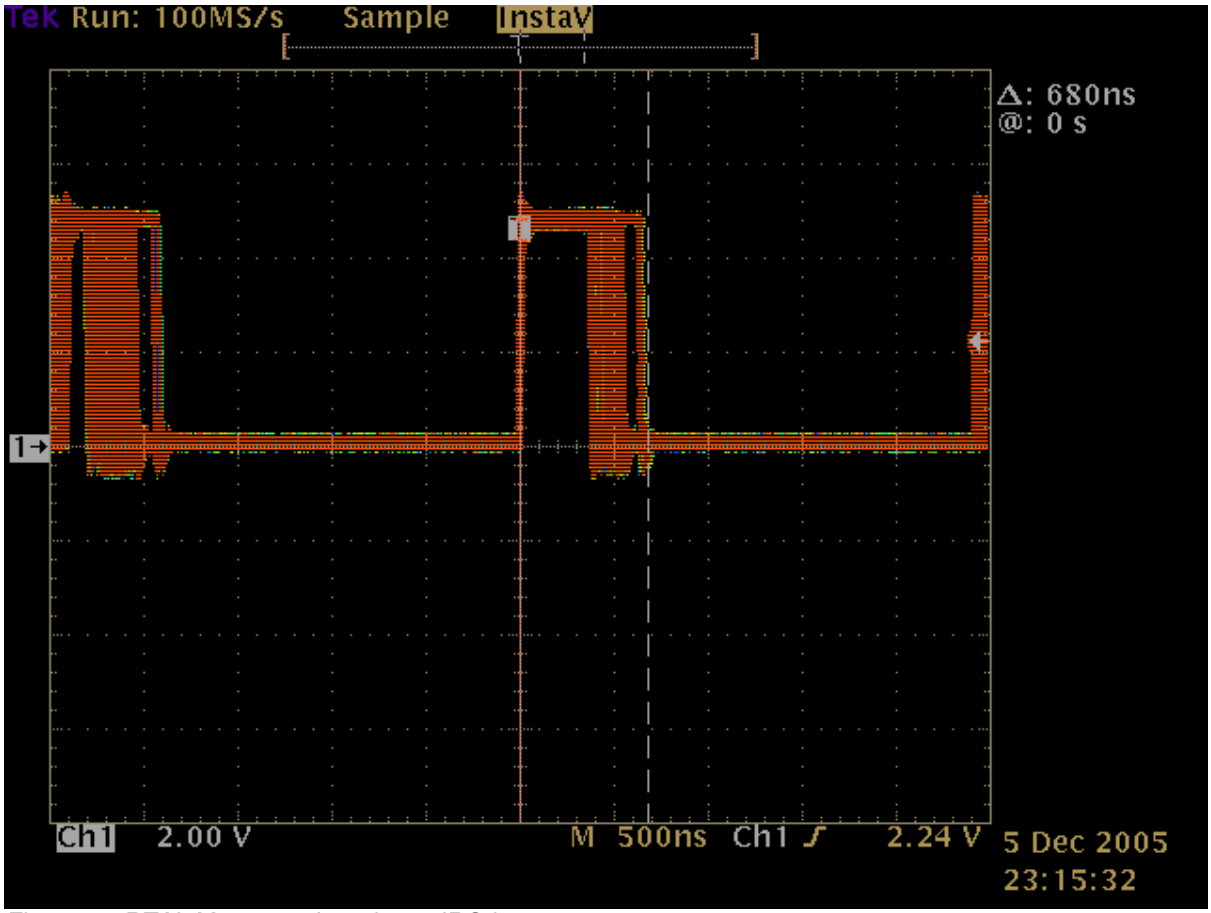


Figure 1 – RTAI: Measure of maximum IRQ latency

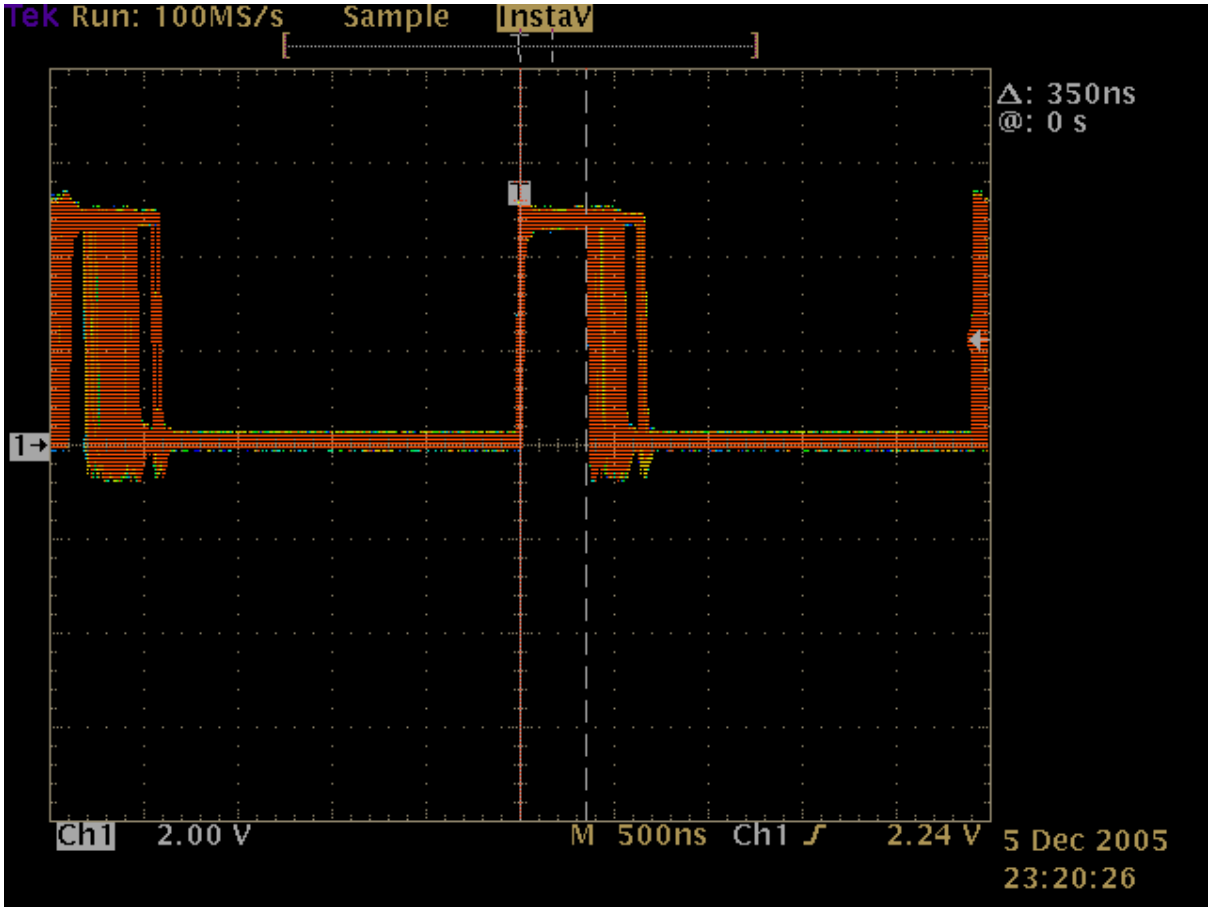


Figure 2 - RTAI: Measure of minimum IRQ latency

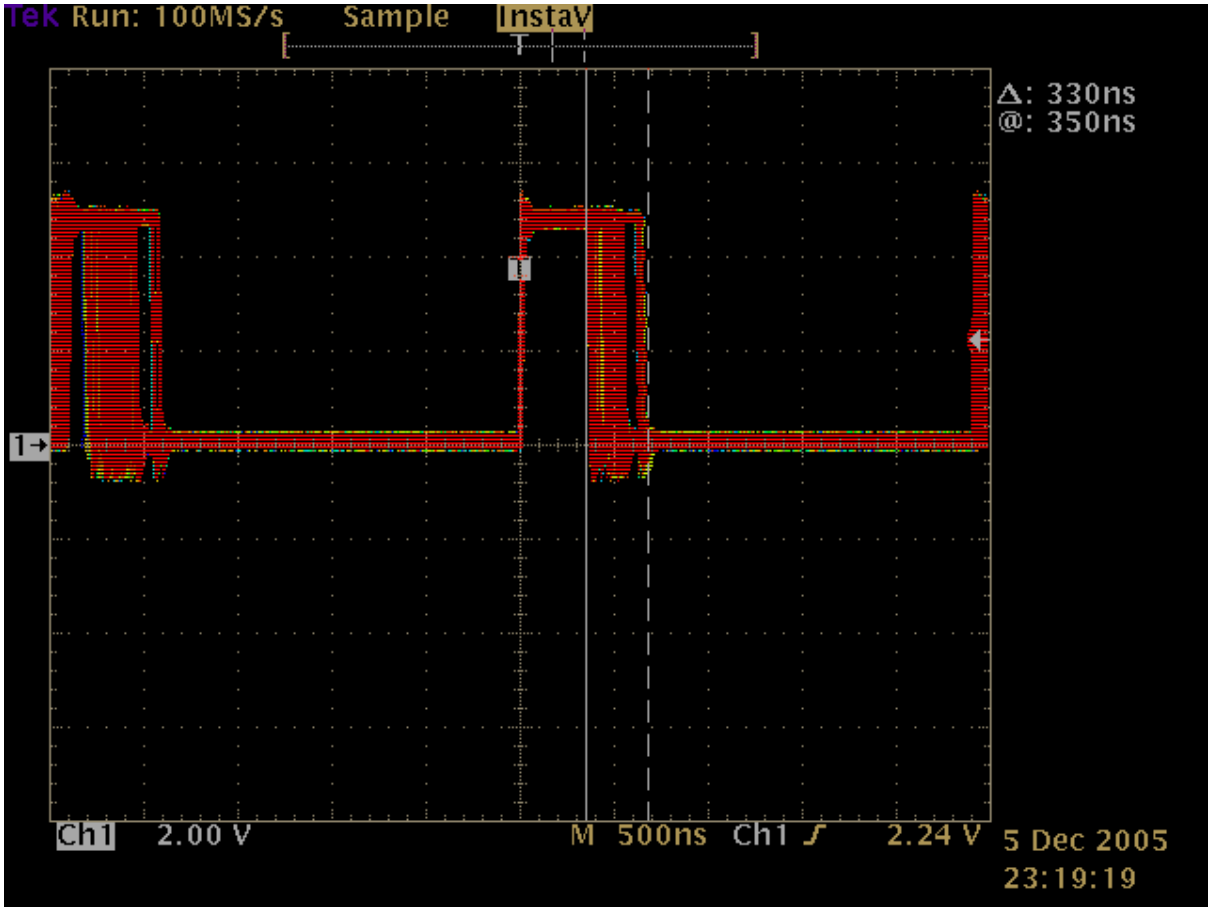


Figure 3 - RTAI: Measure of maximum IRQ variation

2.1.3 RTLinux Measurements

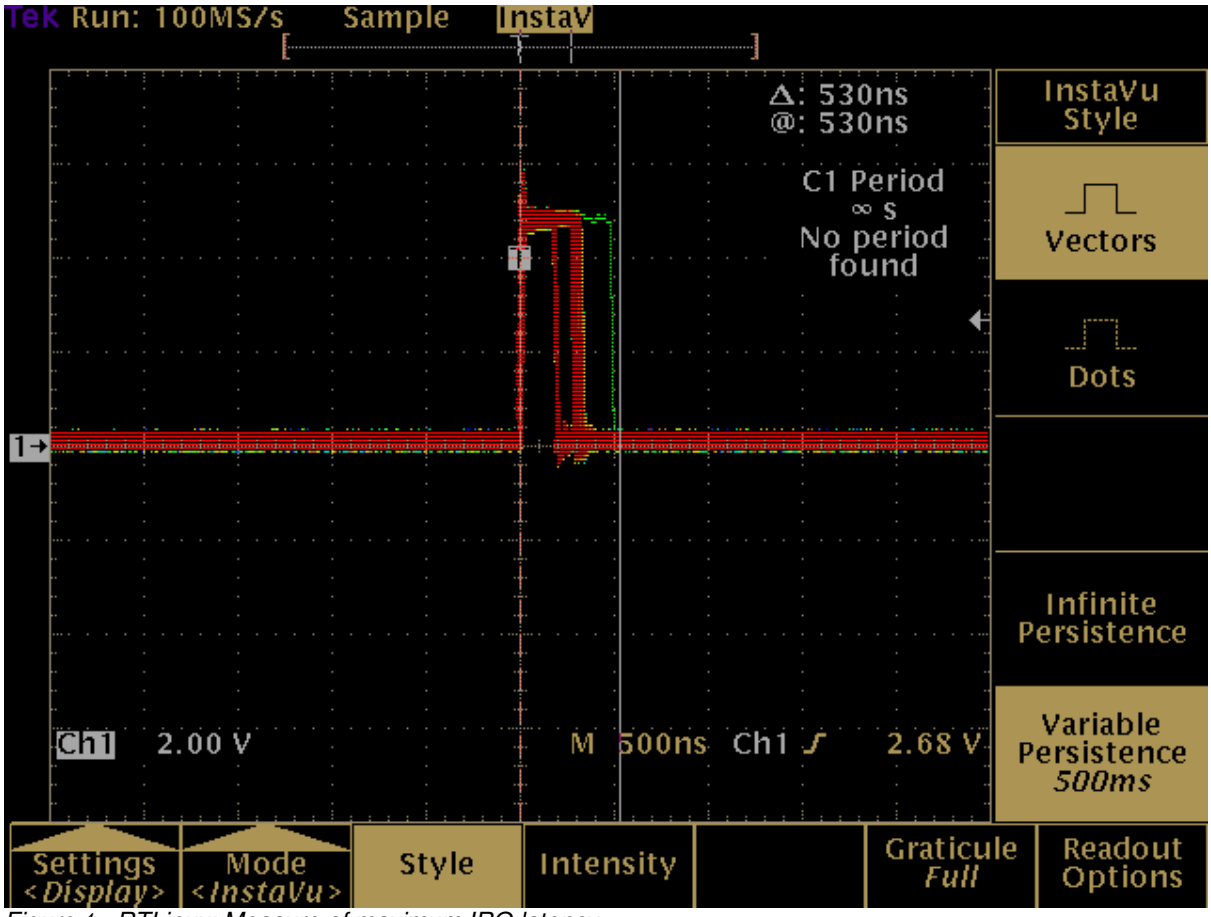


Figure 4 - RTLinux: Measure of maximum IRQ latency

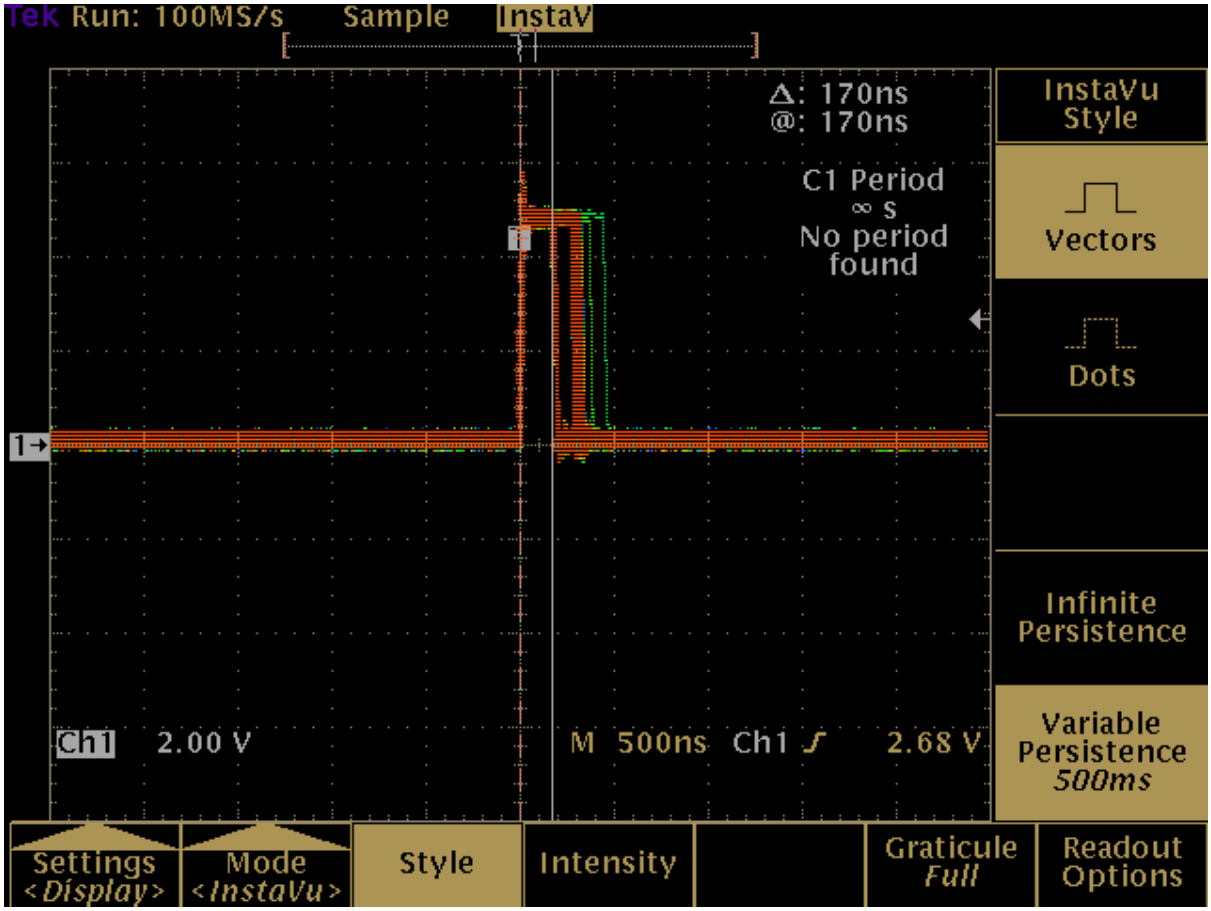


Figure 5 - RTLinux: Measure of minimum IRQ latency

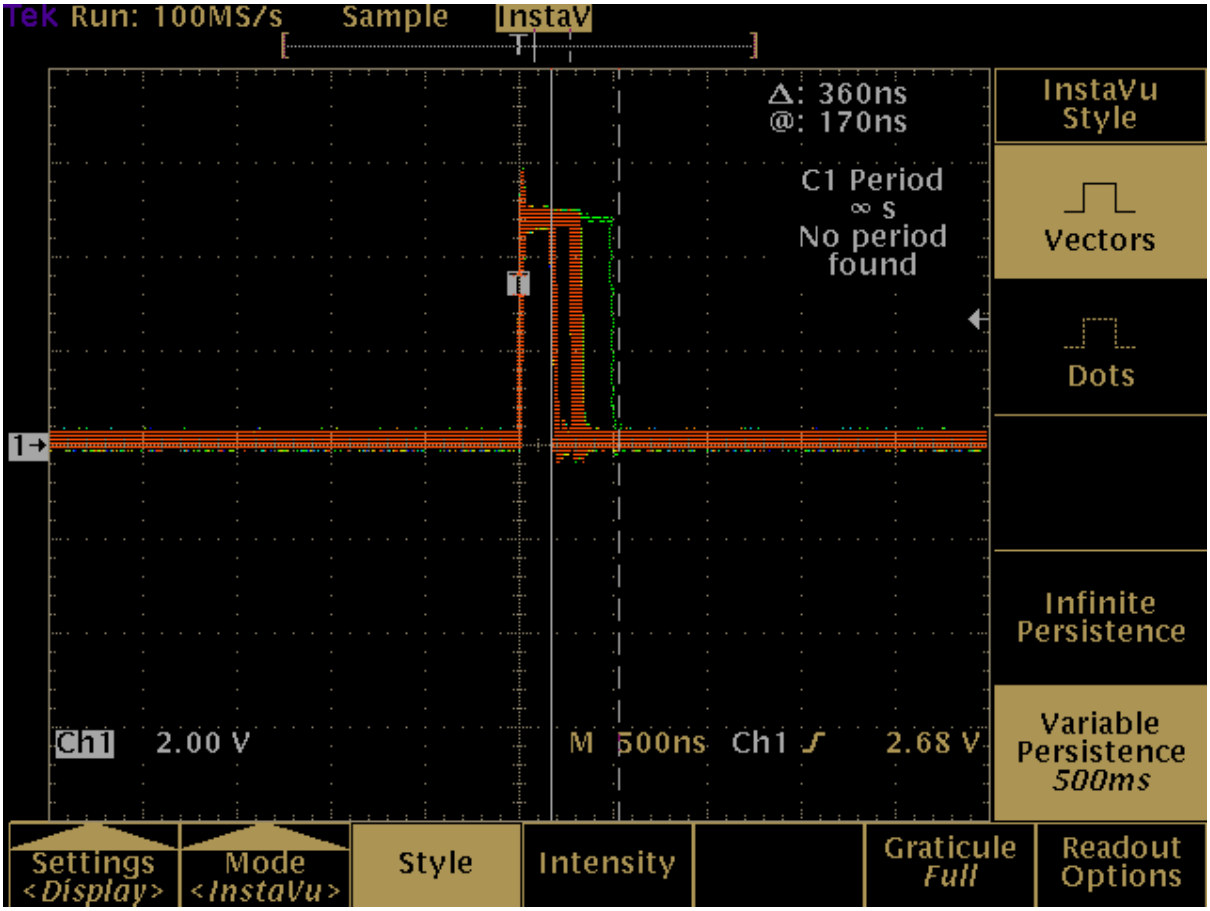


Figure 6 - RTLinux: Measure of maximum IRQ variation

2.1.4 VRTXsa Measurements

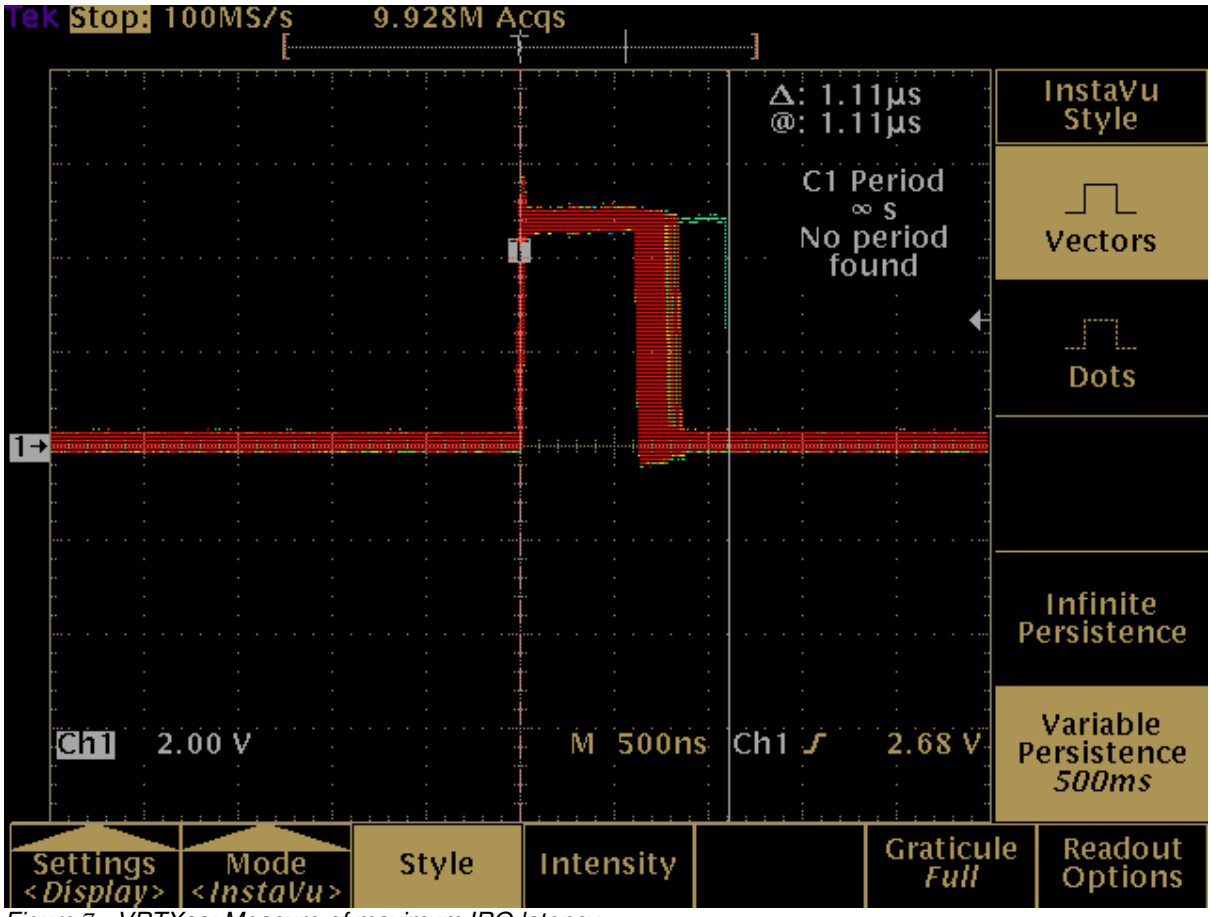


Figure 7 - VRTXsa: Measure of maximum IRQ latency

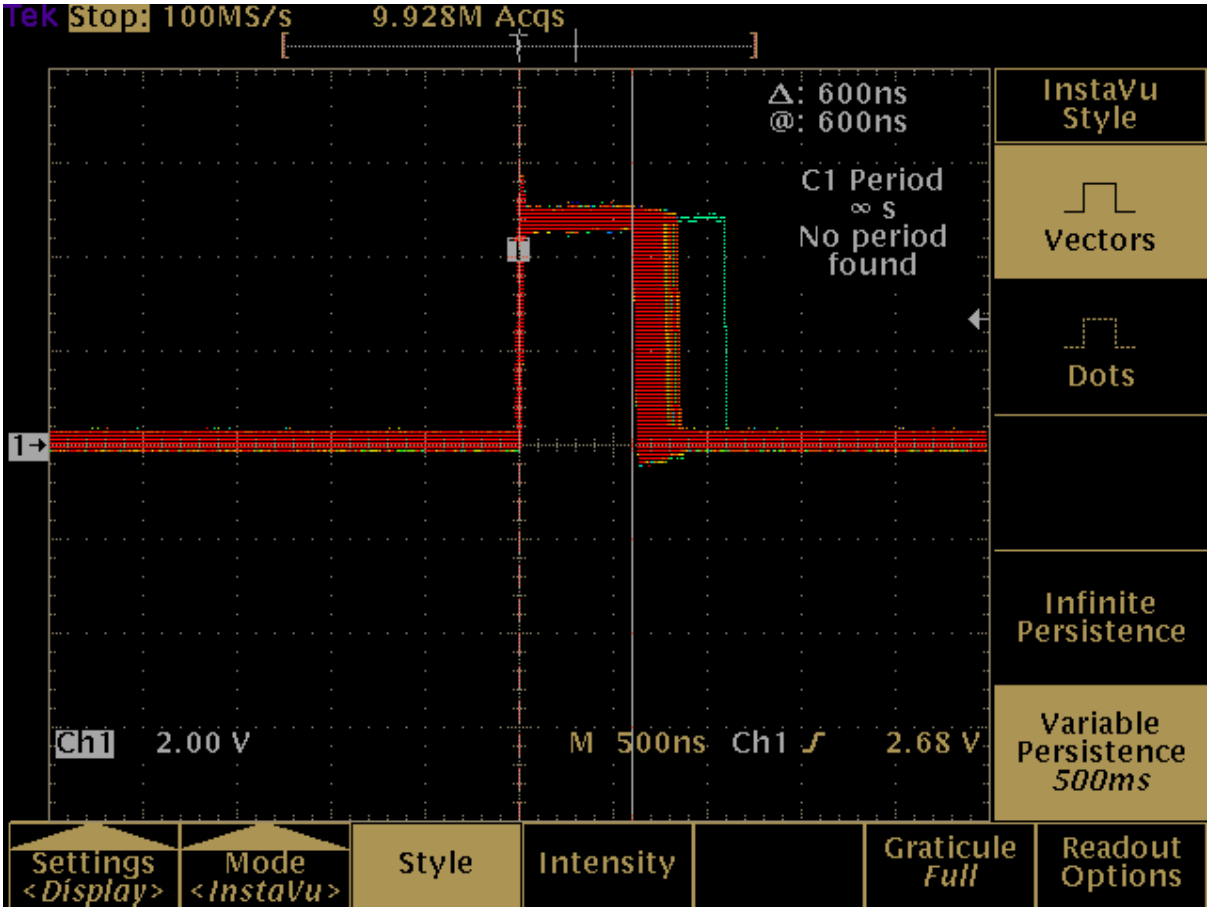


Figure 8 - VRTXsa: Measure of minimum IRQ latency

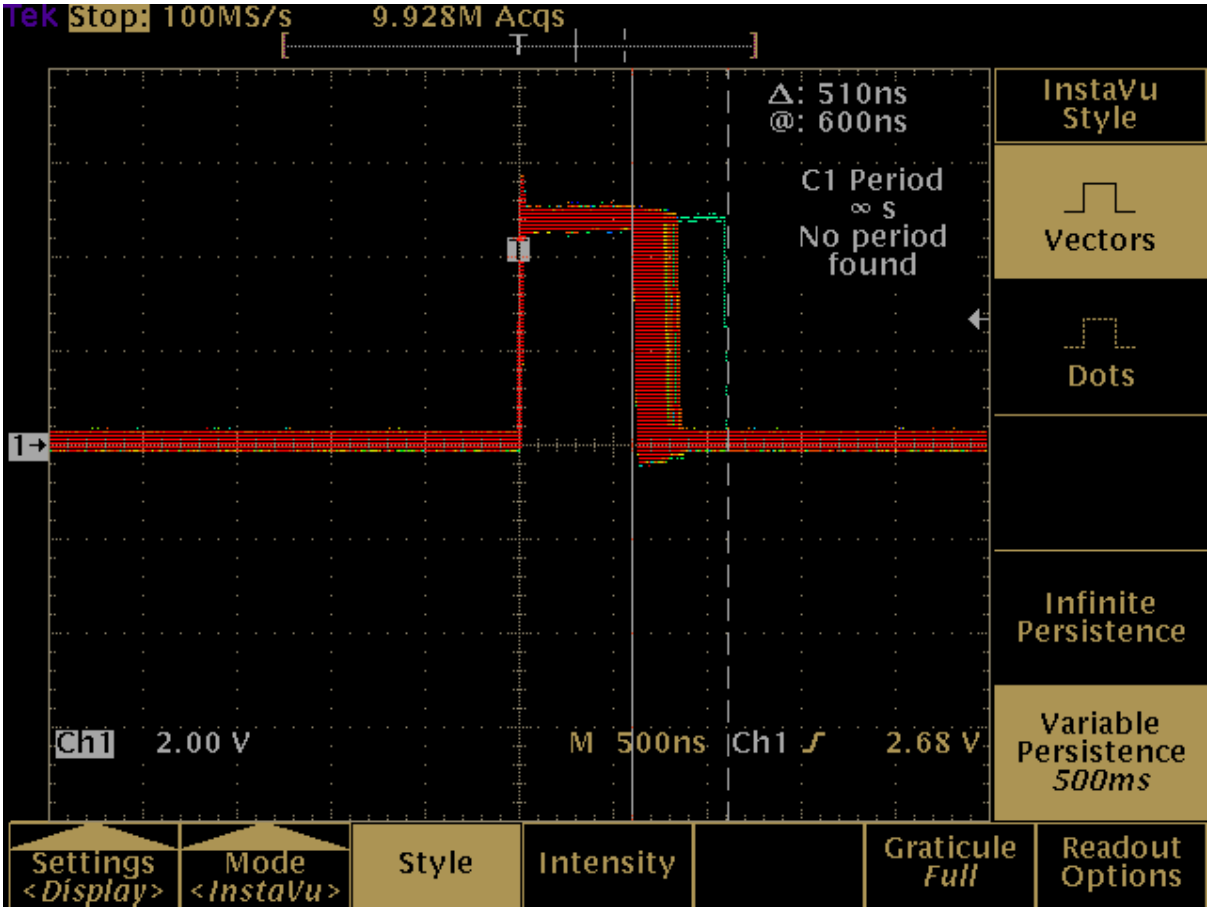
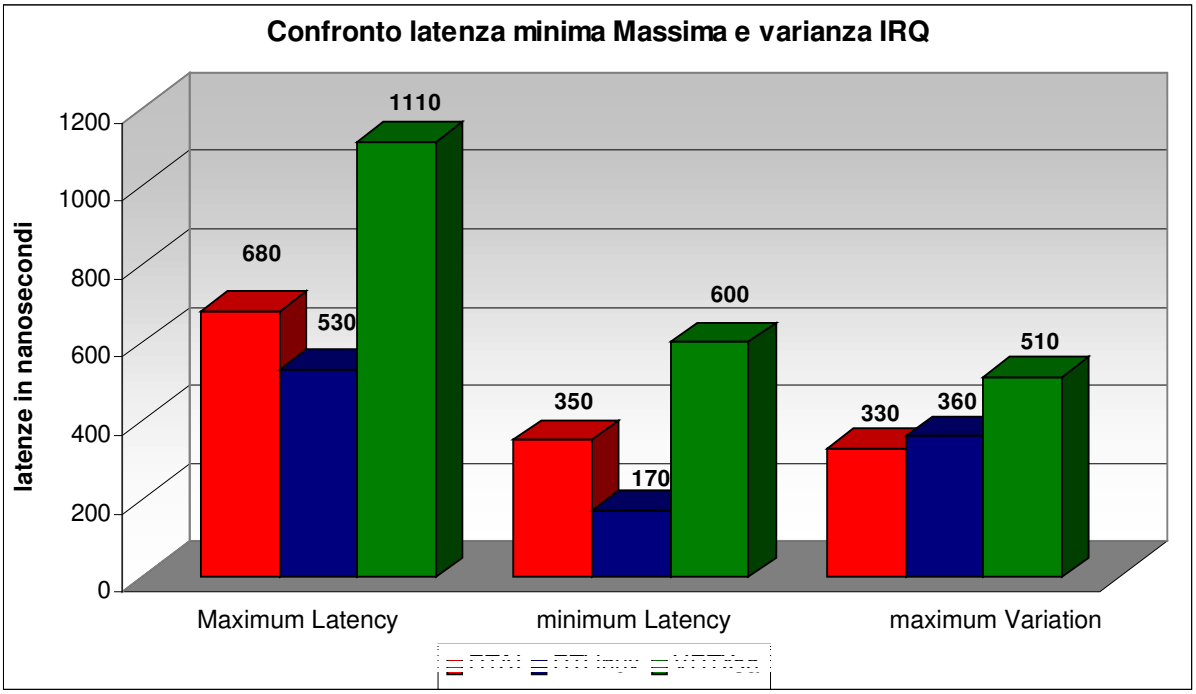


Figure 9 - VRTXsa: Measure of maximum IRQ variation

2.1.5 Results analysis

We compiled the following graph to understand and analyse the data collected:



In a RTOS Variation should be minimized. The difference between the quickest execution and the slowest one has to tend to zero to have a predictable system.
RTAI is the best system with a value of 330 nanoseconds. RTLinux is 30 nanoseconds worst while VRTXsa reports even worst values overstepping 500 nanoseconds. A so high value in variation is given by some executions which raise the maximum latency value over one microsecond. The quickest executions instead report a time around the 600 nanoseconds.
Considering the efficiency VRTXsa surely is the slowest system with a maximum latency higher than a microsecond. RTLinux is the quickest system with a minimum latency lower than 200 nanoseconds.
RTAI reaches intermediate values: it has a slightly more uniform values' set and is able to get the best variation value, 30 nanoseconds lower than RTLinux, being so the most predictable system.

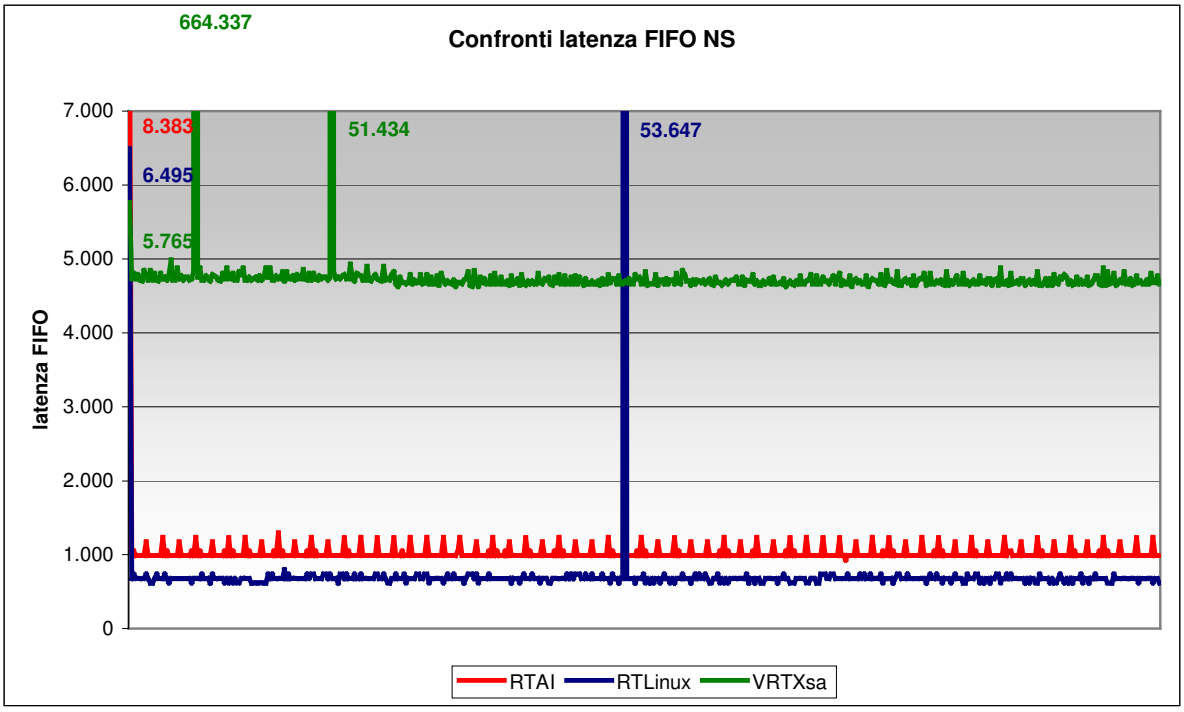
2.2 Software Tests

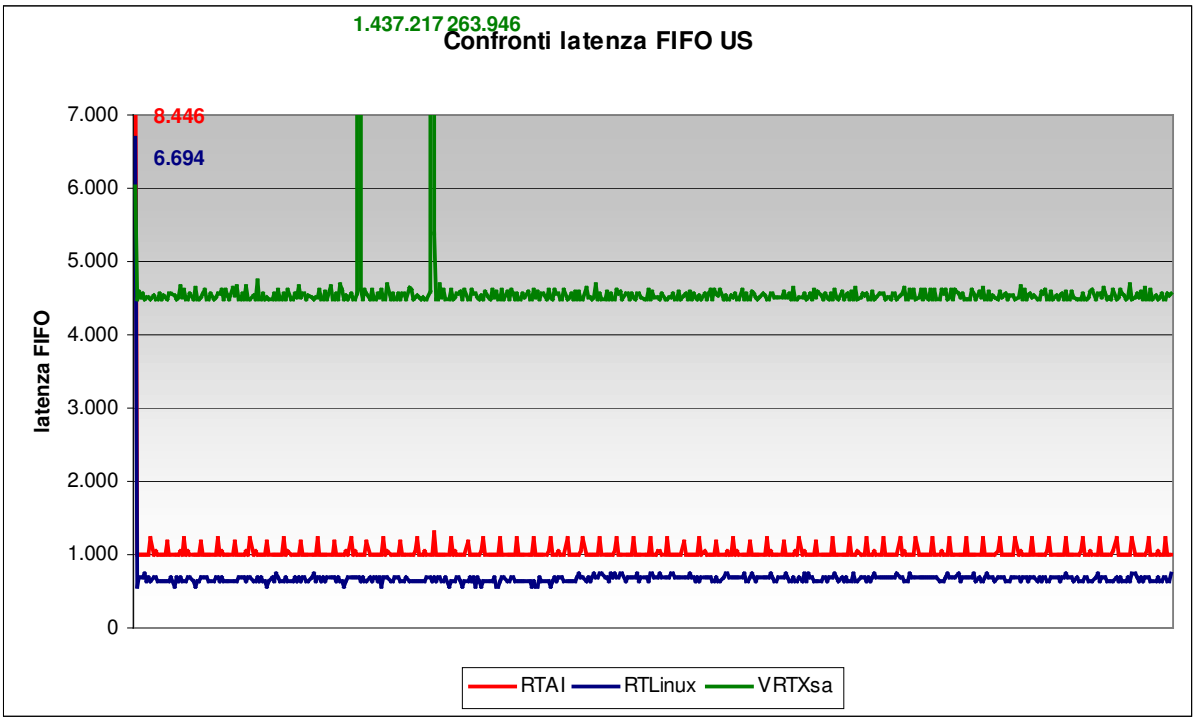
In software tests we measure ipc times using fifo queues and mailboxes.
We're able to measure latency of fifo and mailboxes calls, "intratask" ipc times, and "intertask" ipc times plus context switching. We're able to obtain context switch time too, confronting the previous results.

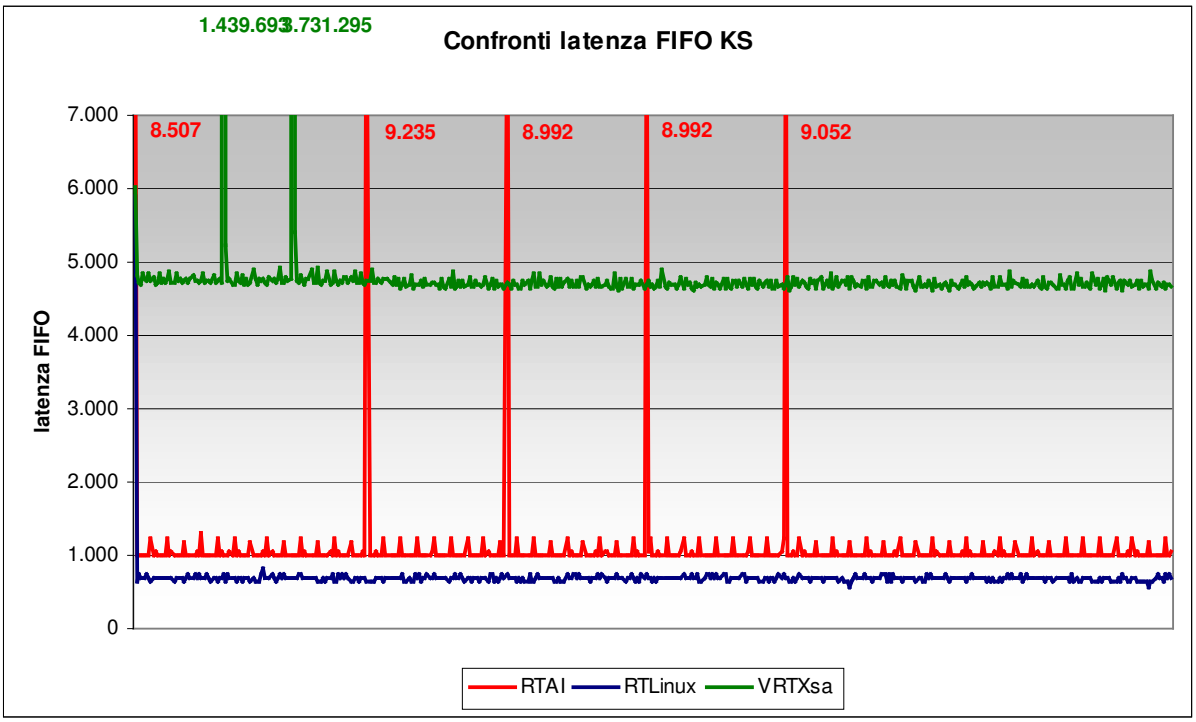
2.2.1 Efficiency

These tests measure the time spent by the operating system to execute some ipc calls.

LatencyFIFO is a test which measures latency times of a call which writes 4 bytes of data on a FIFO queue

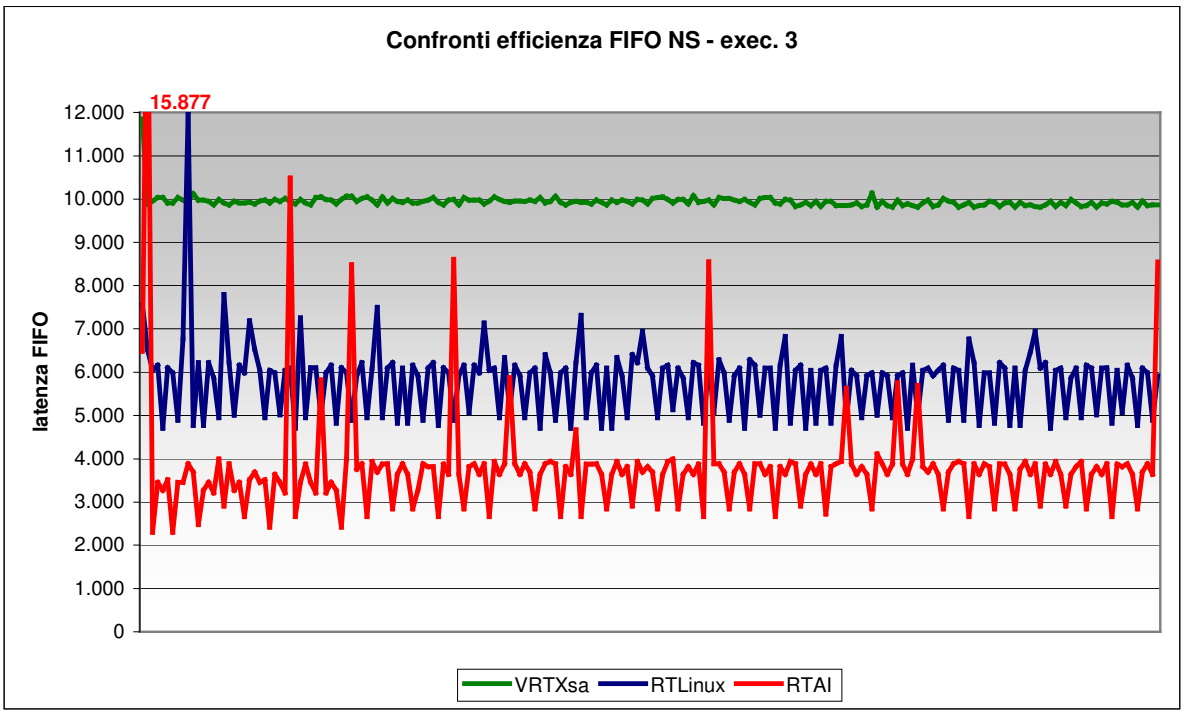


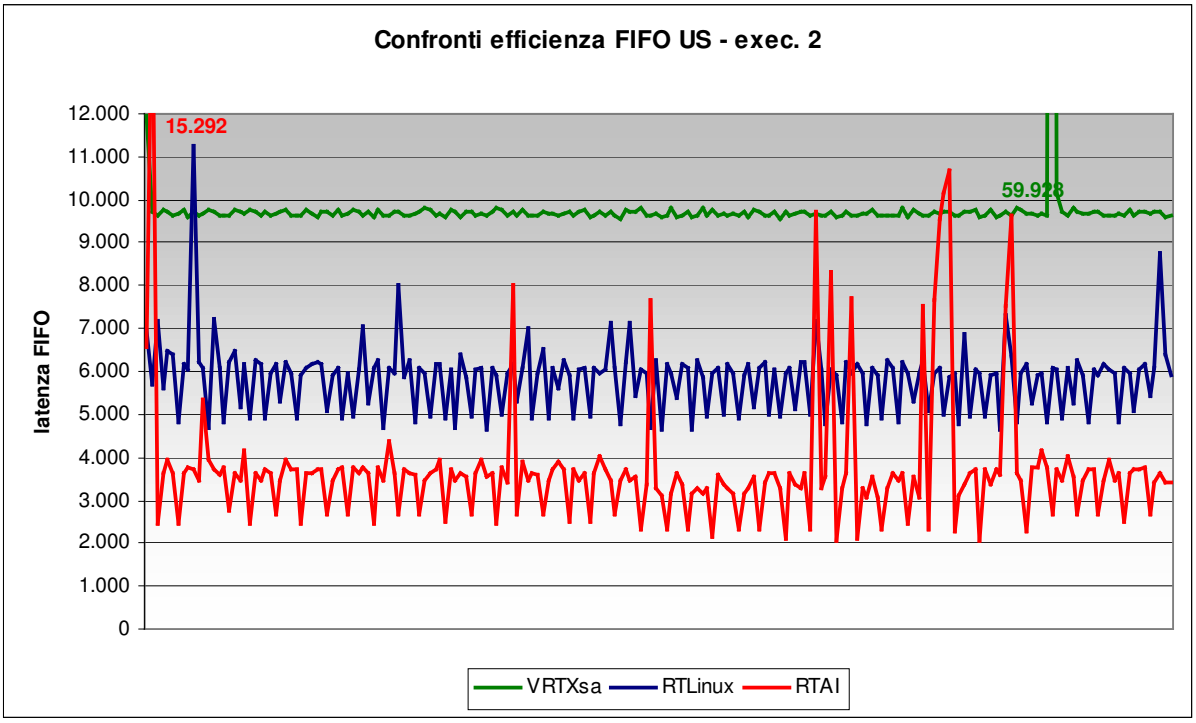


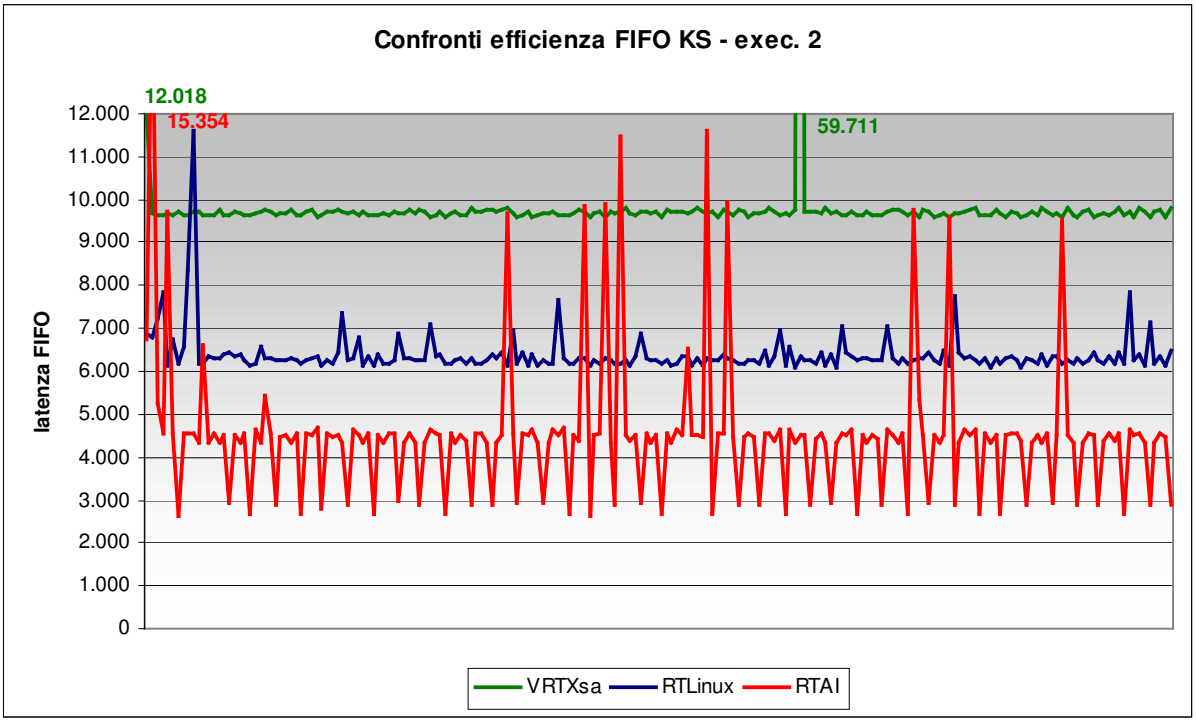


VRTXsa is the slowest system reporting times near 5k nanoseconds. RTAI and RTLinux are much quicker: about 1k nanoseconds for RTAI and about 700 for RTLinux.
These times really are very good and very hard to maintain stable especially under stress, in rare occasions in fact RTAI graph shows some peaks around 9k nanoseconds. VRTXsa graphs don't change very much: some peaks are present in the executions without stress, and these peaks increase in the executions under stress. RTLinux isn't affected by the stress increment and it would have the lowest variation if we don't consider a peak around 53k nanoseconds. This peak is a unique event and may be ignored as an anomaly.

Fifo.c evaluates the time that a task spends to write and read a 4 bytes data block in a fifo queue.

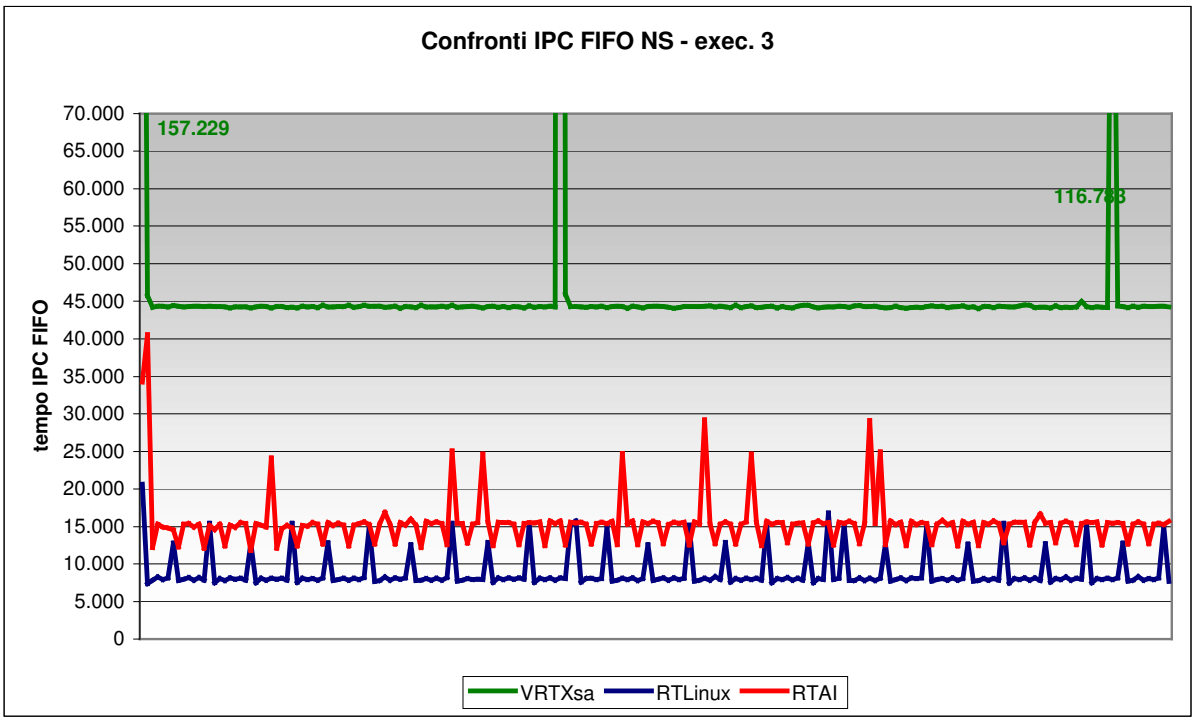

Confronti efficienza FIFO NS - exec. 3
15.877
12.000
11.000
10.000
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0
latenza FIFO
VRTXsa
RTLinux
RTAI

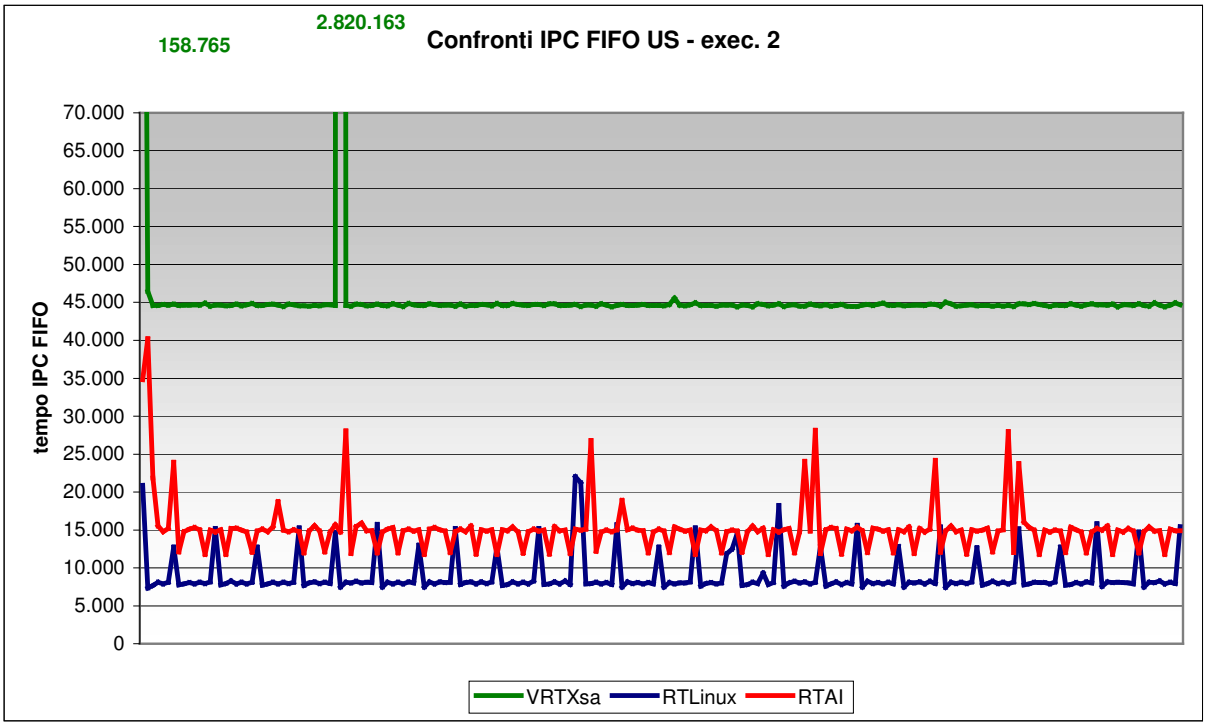

Confronti efficienza FIFO US - exec. 2
15.292
59.928
12.000
11.000
10.000
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0
latenza FIFO
VRTXsa
RTLinux
RTAI

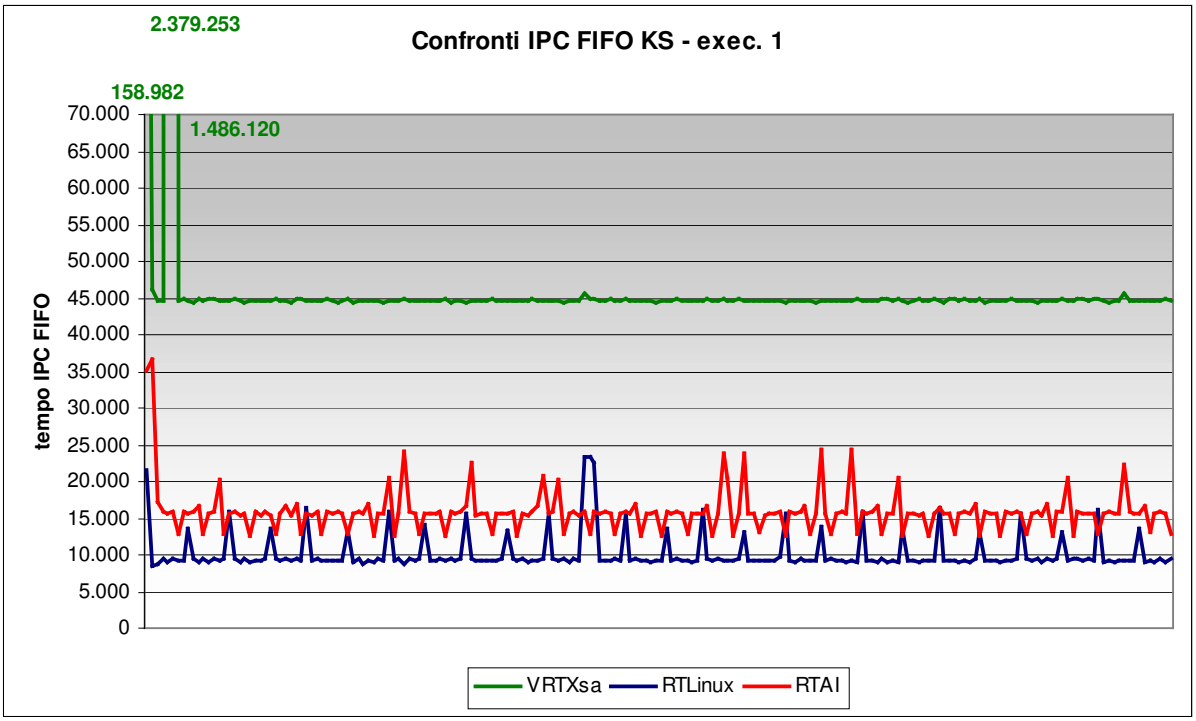


RTAI reports times about 2k nanoseconds lower than RTLinux which has however the less perturbed graph and, consequently, a lower variation than RTAI. The best system, considering the graph perturbation, is VRTXsa, but it's much slower than the other systems.

FifoCS.c runs two concurrent tasks synchronized on a FIFO queue used to exchange messages. The time measured is the sum of writing and reading time from fifo queue, synchronization time and context switch time.
Message is a struct of 16 bytes. VRTsa in this case moves only the address pointing to the struct, so only 4 bytes instead of 16. We have to consider this advantage while evaluating the following graphs.

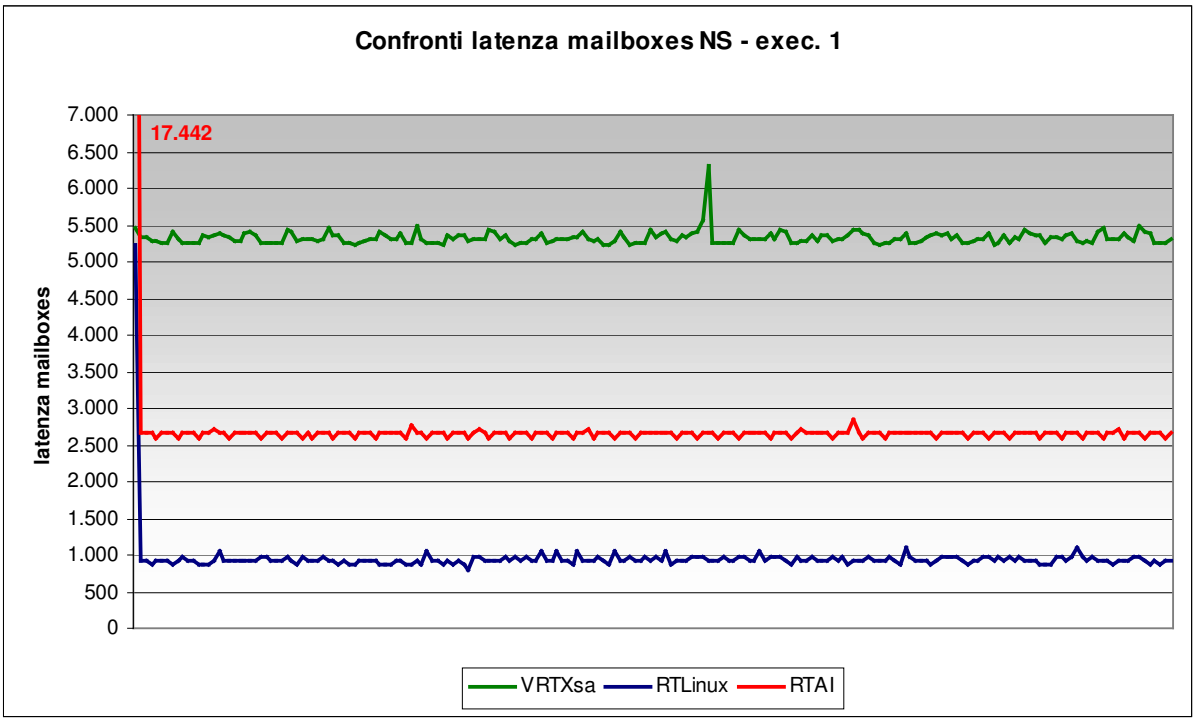


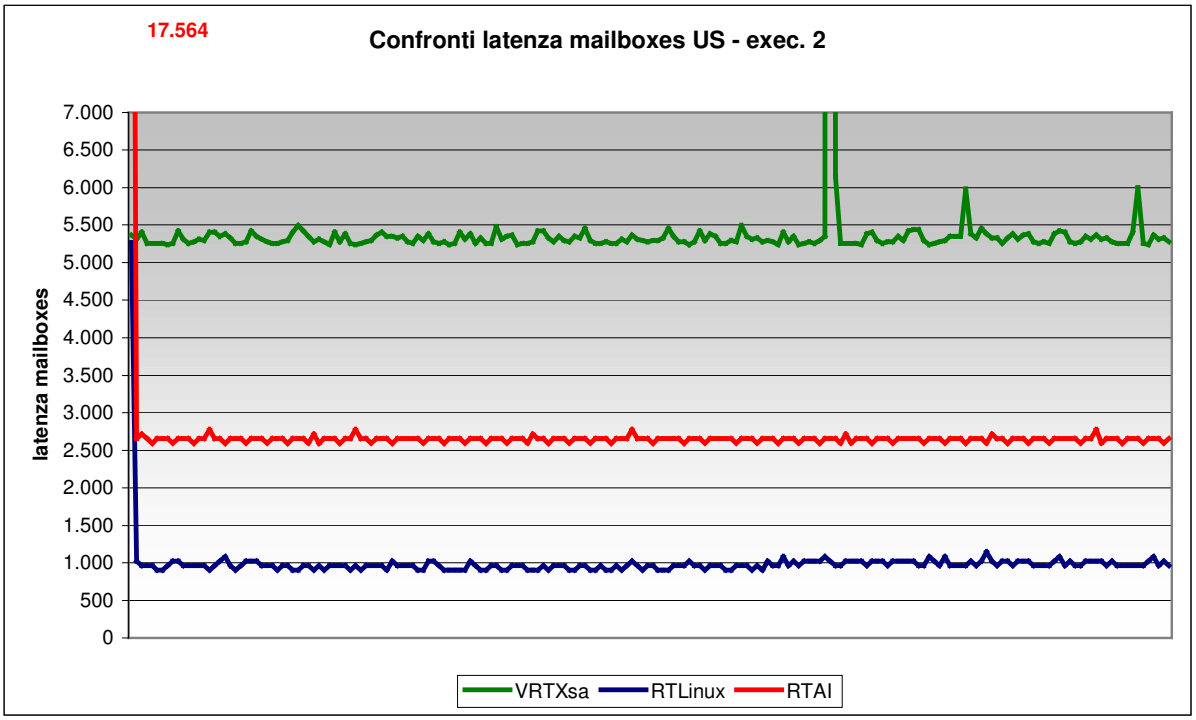


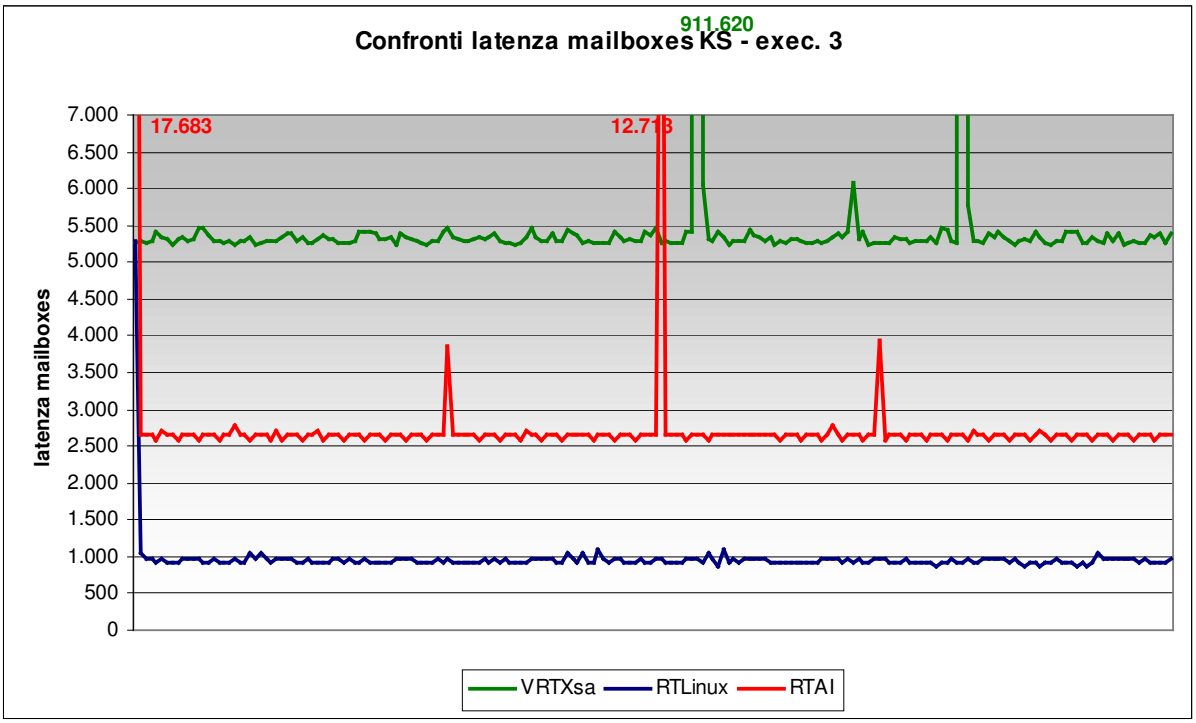


VRTXsa is still the slowest system but with less perturbed graph. It seem it isn't affected by the stress increment: it shows peaks in all three executions although with different values.
Linux systems are more similar considering both variation and efficiency. Also in this case efficiency is constant if stress increases.

mbox.c runs two distinct tests: the first one measures the latency of a call writing a 16 bytes data block to a mailbox.

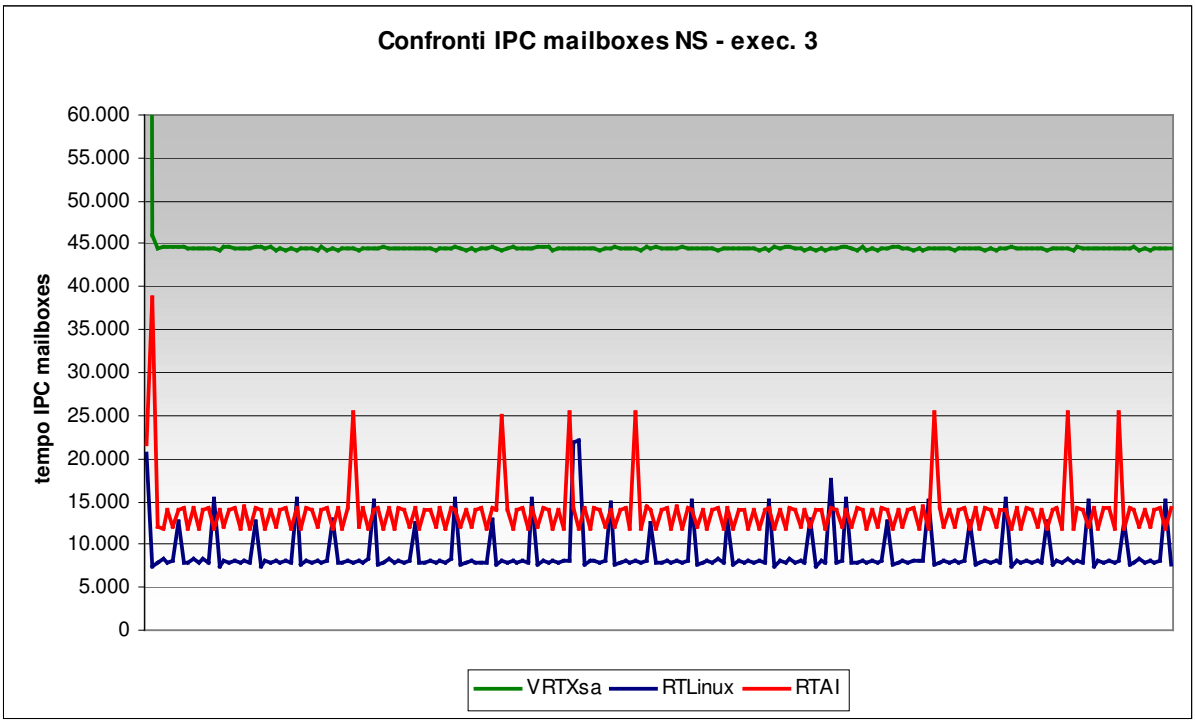

Confronti latenza mailboxes NS - exec. 1
17.442
latenza mailboxes
7.000
6.500
6.000
5.500
5.000
4.500
4.000
3.500
3.000
2.500
2.000
1.500
1.000
500
0
VRTXsa
RTLinux
RTAI

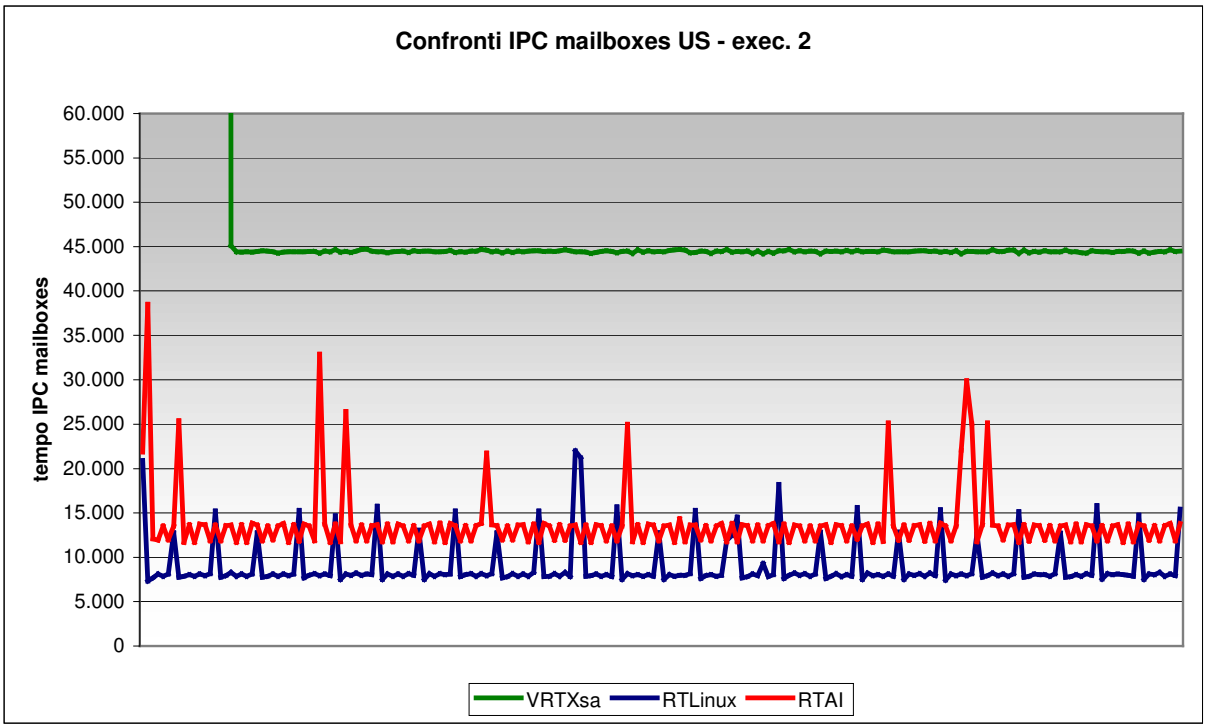

17.564
Confronti latenza mailboxes US - exec. 2
latenza mailboxes
7.000
6.500
6.000
5.500
5.000
4.500
4.000
3.500
3.000
2.500
2.000
1.500
1.000
500
0
VRTXsa
RTLinux
RTAI

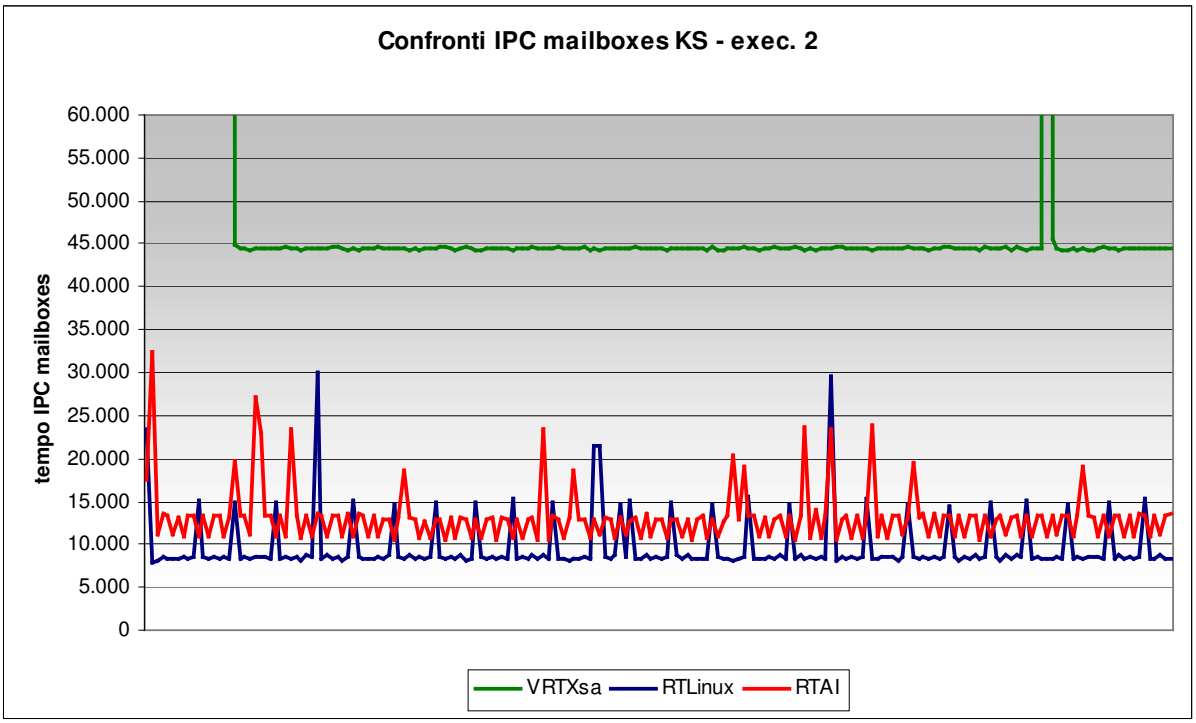


RTLinux seems to be the quickest system and VRTXsa the slowest one.
RTAI is in the middle. RTAI's graph seems to be the less perturbed followed by RTLinux and VRTXsa which report very high peaks.
RTAI seems to be much slower with stress in kernel space: it has a more perturbed graph. RTLinux instead is the system which isn't affected by the stress increment. Efficiency is constant for all the three systems but VRTXsa variation increases very much as RTAI variation increases a little.

The second test is a mailboxes ipc test: two concurrent tasks exchange a message using a mailbox.
In this case time is the sum of writing and reading from mailbox, synchronization time and context switch time. Message is a struct of a 16 bytes data block. VRTXsa moves only the addresses of this messages, so only 4 bytes instead of 16. We have to consider this advantage while evaluating the following graphs.


Confronti IPC mailboxes NS - exec. 3
tempo IPC mailboxes
60.000
55.000
50.000
45.000
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
VRTXsa
RTLinux
RTAI

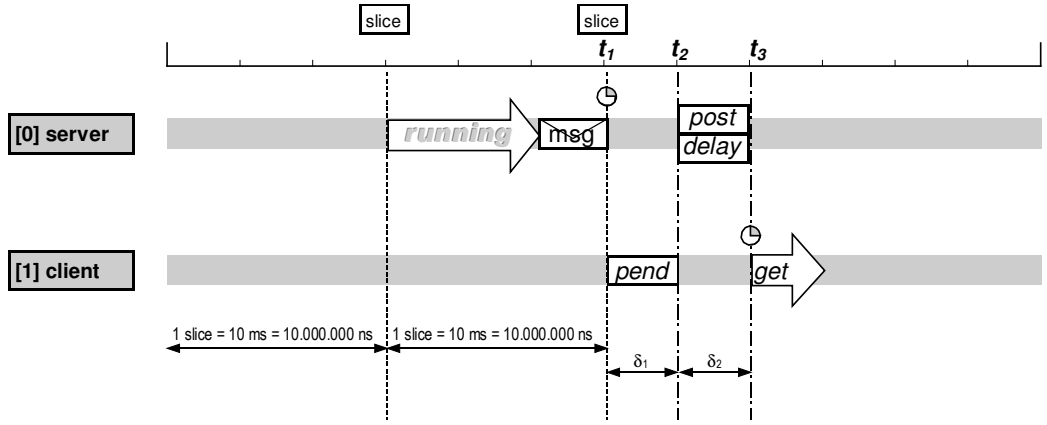

Confronti IPC mailboxes US - exec. 2
tempo IPC mailboxes
60.000
55.000
50.000
45.000
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
VRTXsa
RTLinux
RTAI



RTLinux is the quickest system in this case too and isn't affected by the stress increment considering both variarion and efficiency. RTAI shows some peaks and a more perturbed graph if stress is increased.
VRTXsa is the slowest system but has the less perturbed graph. It reports some peak which cause a high variation value.
But we have to analyse better VRTXsa's behaviour. In the executions under stress a big number of iterations reported values of about 10 milliseconds. In some cases it didn't happened and in rare cases the whole execution of the test reported these values.
In some cases the concurrent VRTXsa tasks are synchronized in a bad way and a whole system tick (just 10 milliseconds) elapses between the composition of the message and its receiving.
This execution schema may explain better the idea:

DIAGRAMMI DI FLUSSO DI ESECUZIONE caso 1

OS: VRTXsa sorgente: mbox.c tipo di carico: assente



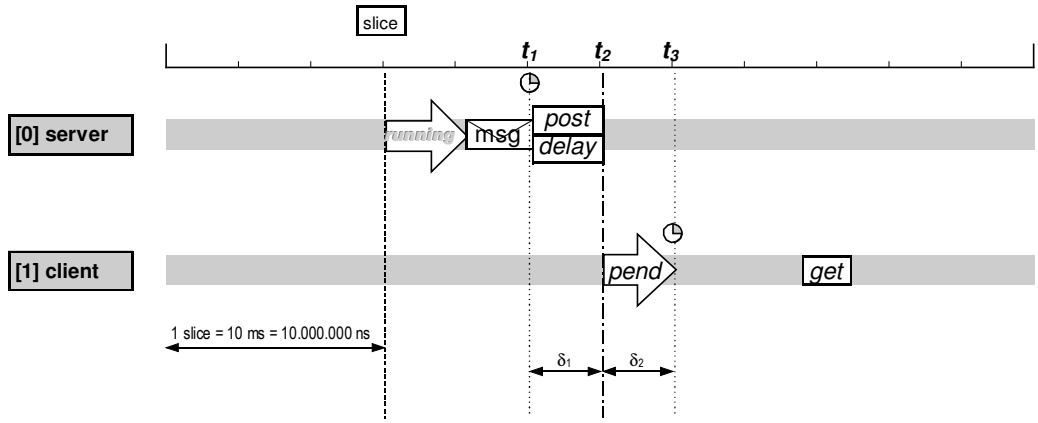
Il test calcola la differenza di tempo (in ns) che intercorre tra l'istante in cui il server crea il messaggio e l'istante in cui il client lo riceve.

Un messaggio inviato ad una mailbox con *sc_post* è immediatamente allocato ad un eventuale task che è in attesa sulla mailbox. Il messaggio non è salvato nella mailbox in questo caso. Questo comportamento di VRTXsa fa in modo che al tempo t_3 il client abbia in realtà già ricevuto il messaggio che stava aspettando.
Dunque si avrà che:

Tempo IPC: $(t_3 - t_1) = (\delta_1 + \delta_2) =$ ca. 45000 ns

DIAGRAMMI DI FLUSSO DI ESECUZIONE caso 2

OS: VRTXsa sorgente: mbox.c tipo di carico: assente



Il test calcola la differenza di tempo (in ns) che intercorre tra l'istante in cui il server crea il messaggio e l'istante in cui il client lo riceve.

Dunque si avrà che:

Tempo IPC: $(t_3 - t_1) = (\delta_1 + \delta_2) =$ ca. 45000 ns

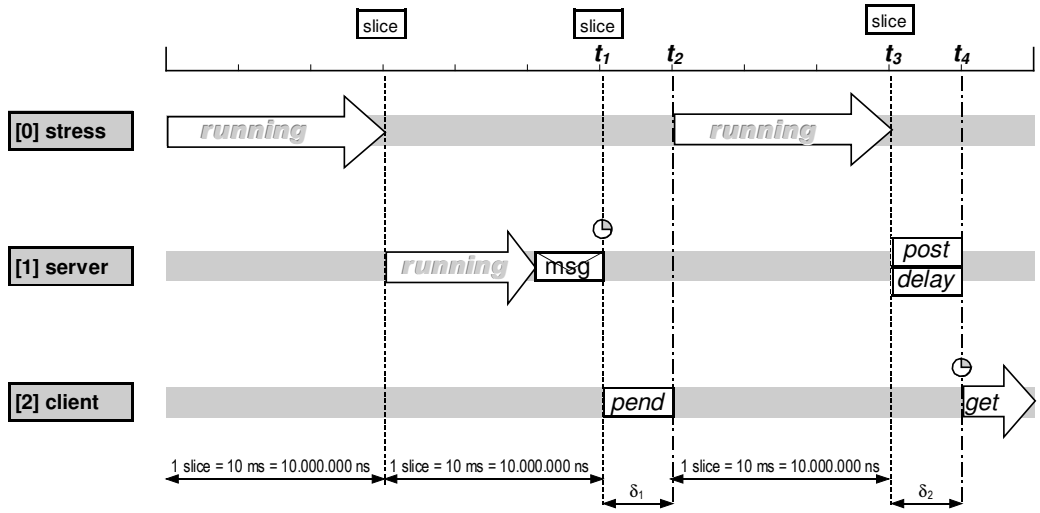
Such schemas show the normal synchronization possible between client task and server task in an execution without stress. In both cases at instant t1 message is composed by the server and at instant t3 message is read by the client.

In a stressed execution the scheduler has to manage three tasks.
It's important to remember that these tasks are scheduled with a round robin time slice policy with a static fifo priority. Time slice can be set only to an integer multiple of the system tick, so minimum value is just one system tick = 10 milliseconds.
Stress task is the first task to be executed. When its slice is over scheduler will bring it to ready state and will run instead the server task. It may happens that its slice ends after server composed the message and just before it writes it on the mailbox, so message contains value t1.
Now scheduler will run client task which will execute immediately an sc_pend() call on mailbox. This call is blocking, so rescheduling procedure is started and scheduler will run the stress task according to fifo policy. Stress task will run for a time slice. After this period the server task will run again and it will write the message on the mailbox and then will wait on the synchronization semaphore.
Since client task was already waiting for the message, VRTXsa will move the message right from the server to the client.
So the elapsed time now will be the time between instant t4 and t1 and will comprehend a whole time slice: about 10 milliseconds.

The following schema tries to show what just described:

DIAGRAMMI DI FLUSSO DI ESECUZIONE

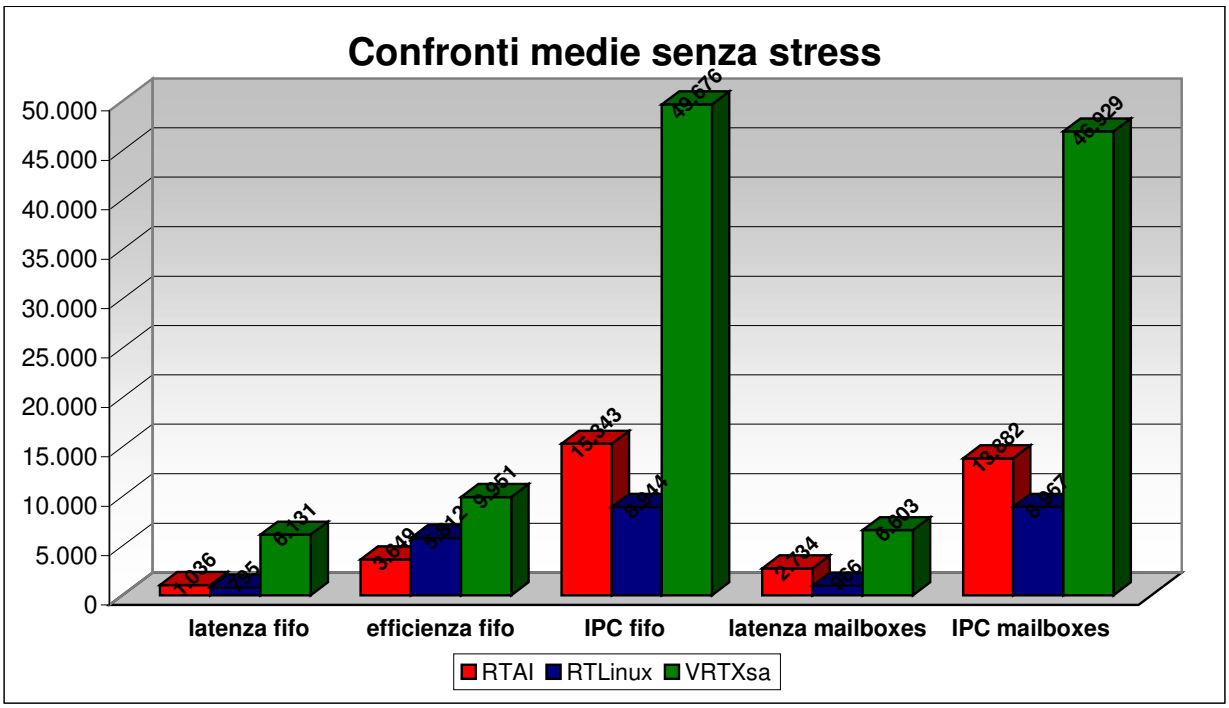
OS: VRTXsa sorgente: mbox.c tipo di carico: kernel mode

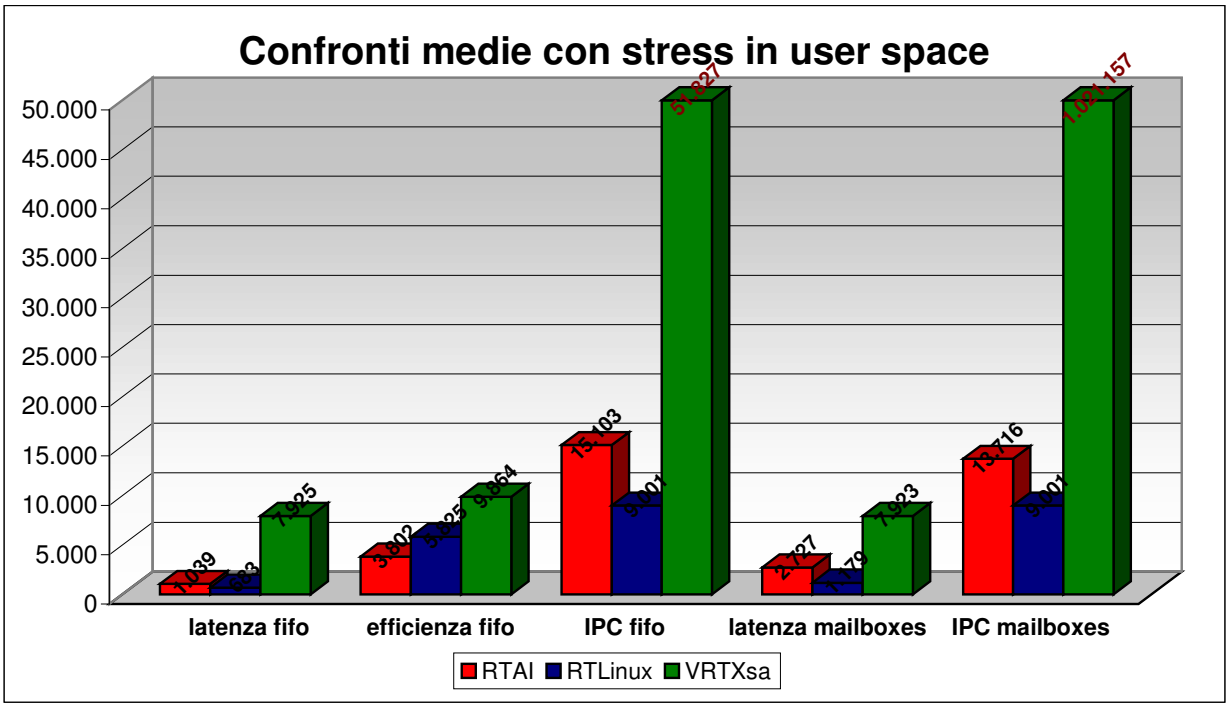


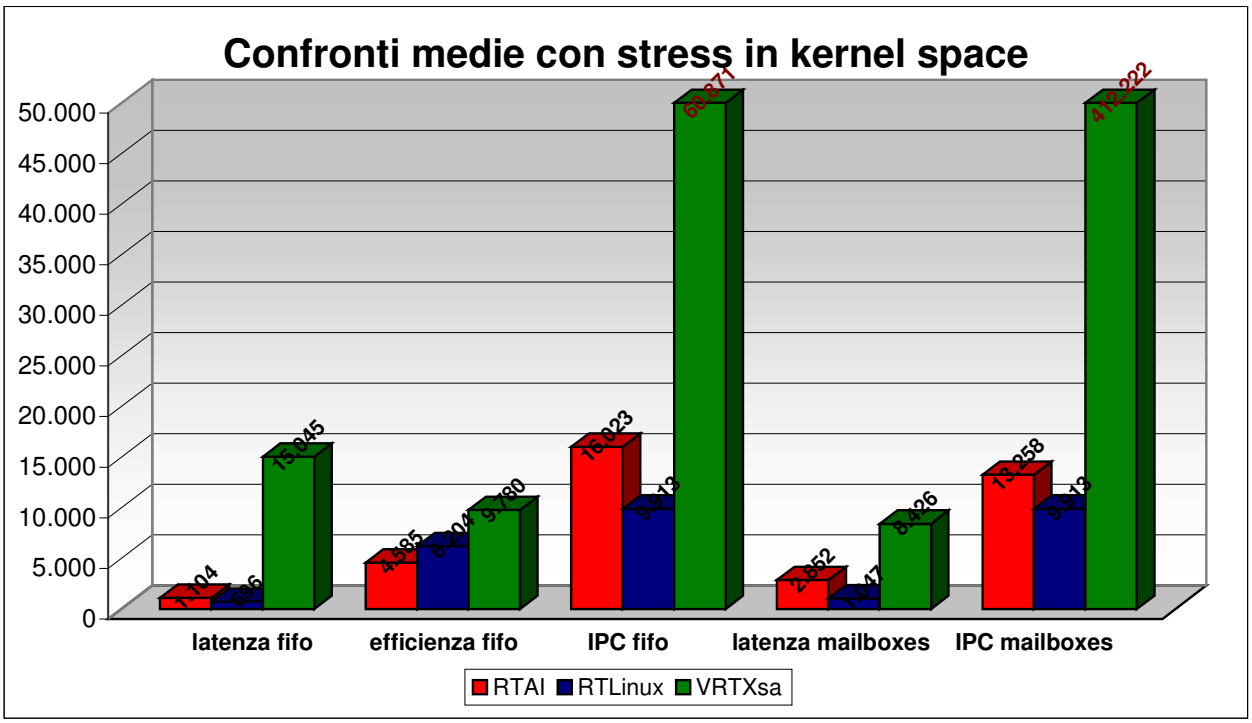
Il test calcola la differenza di tempo (in ns) che intercorre tra l'istante in cui il server crea il messaggio e l'istante in cui il client lo riceve.

Un messaggio inviato ad una mailbox con *sc_post* è immediatamente allocato ad un eventuale task che è in attesa sulla mailbox. Il messaggio non è salvato nella mailbox in questo caso. Questo comportamento di VRTXsa fa in modo che al tempo t_4 il client abbia in realtà già ricevuto il messaggio che stava aspettando.
Dunque si avrà che:

$$\text{Tempo IPC: } (t_4 - t_1) = (1 \text{ slice} + \delta_1 + \delta_2) = \text{ca. } 10000000 \text{ ns}$$

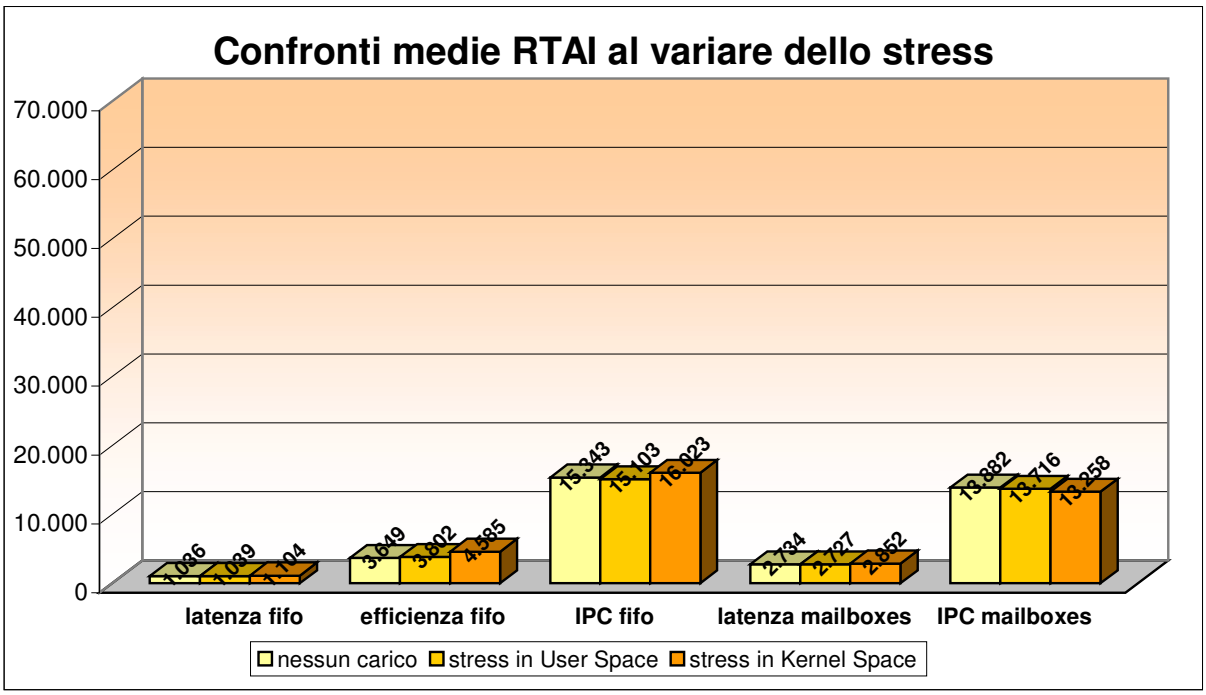

Confronti medie senza stress
50.000
45.000
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
latenza fifo
efficienza fifo
IPC fifo
latenza mailboxes
IPC mailboxes
RTAI
RTLinux
VRTXsa

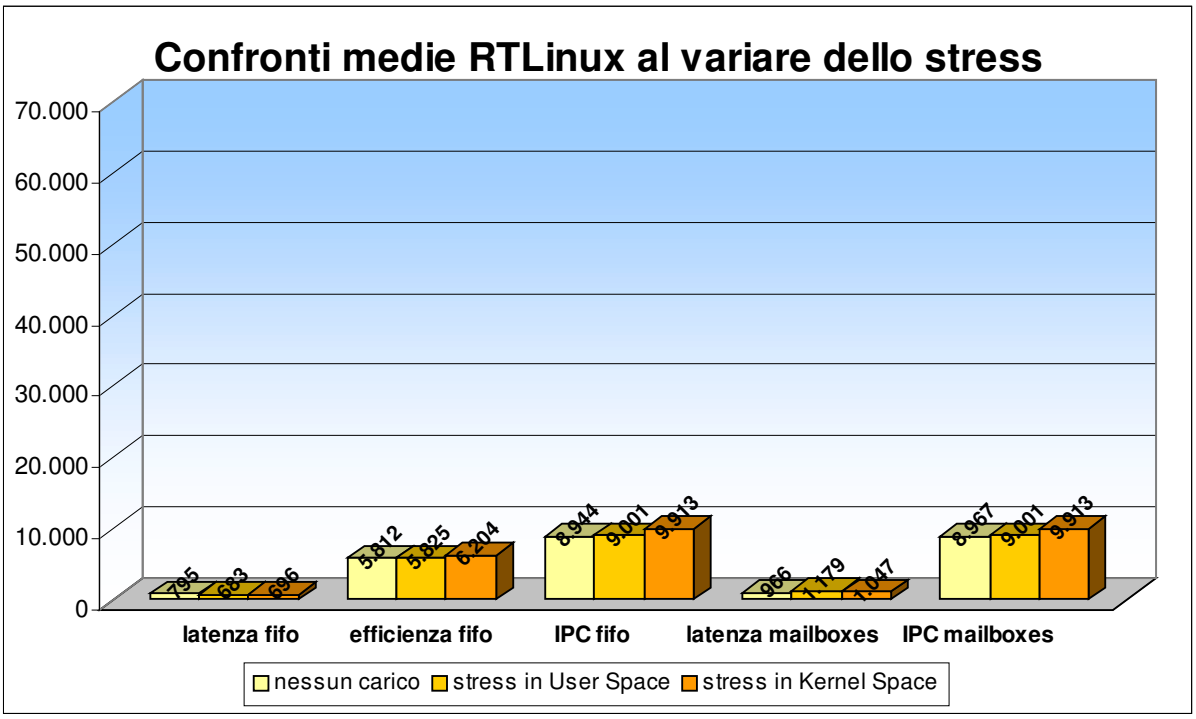

Confronti medie con stress in user space
50.000
45.000
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0
latenza fifo
efficienza fifo
IPC fifo
latenza mailboxes
IPC mailboxes
RTAI
RTLinux
VRTXsa

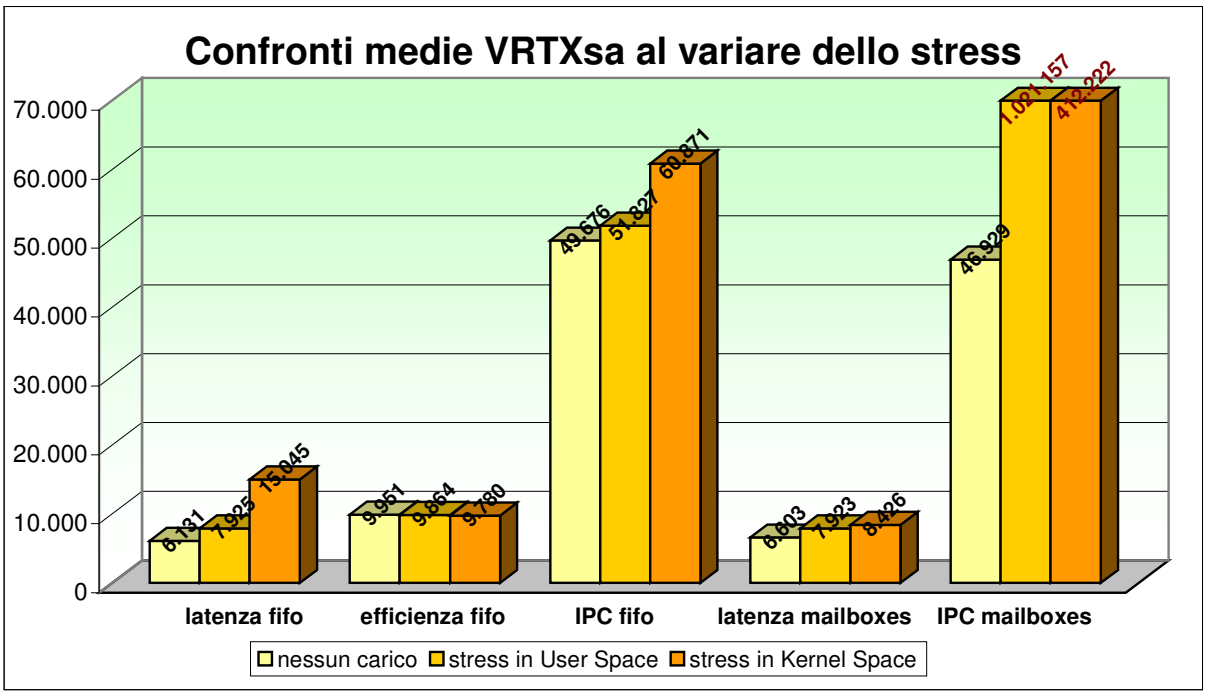


We realized these graph to confront all the average values reported by every operating system in different tests. It's evident the difference between VRTXsa, generally the slowest system, and linux based systems. RTLinux seems to be generally the quickest system.

The same values have been posed in relationship with stress to see immediately if there is influence of stress on the performances of a system. What is clear is that RTAI and RTLinux aren't affected by the stress increment, while VRTXsa is affected in an evident way and proportionally to the kind of stress, especially in ipc tests.

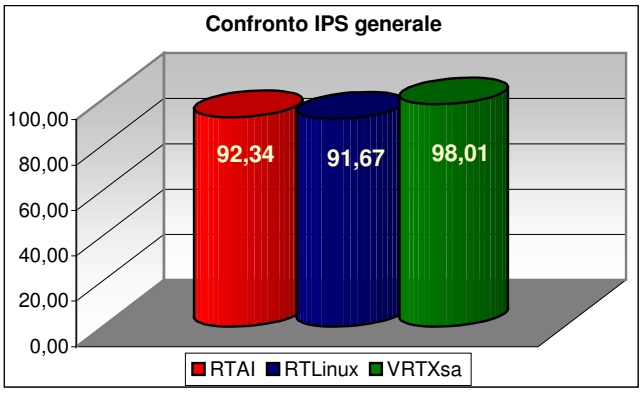

Confronti medie RTAI al variare dello stress
70.000
60.000
50.000
40.000
30.000
20.000
10.000
0
1.086
1.089
1.104
3.649
3.802
4.385
15.343
13.103
16.023
2.734
2.727
2.852
13.882
13.716
13.258
latenza fifo
efficienza fifo
IPC fifo
latenza mailboxes
IPC mailboxes
nessun carico
stress in User Space
stress in Kernel Space


Confronti medie RTLinux al variare dello stress
70.000
60.000
50.000
40.000
30.000
20.000
10.000
0
795
683
696
5.812
5.825
6.204
8.944
9.001
9.913
956
1.179
1.047
8.967
9.001
9.913
latenza fifo
efficienza fifo
IPC fifo
latenza mailboxes
IPC mailboxes
nessun carico
stress in User Space
stress in Kernel Space

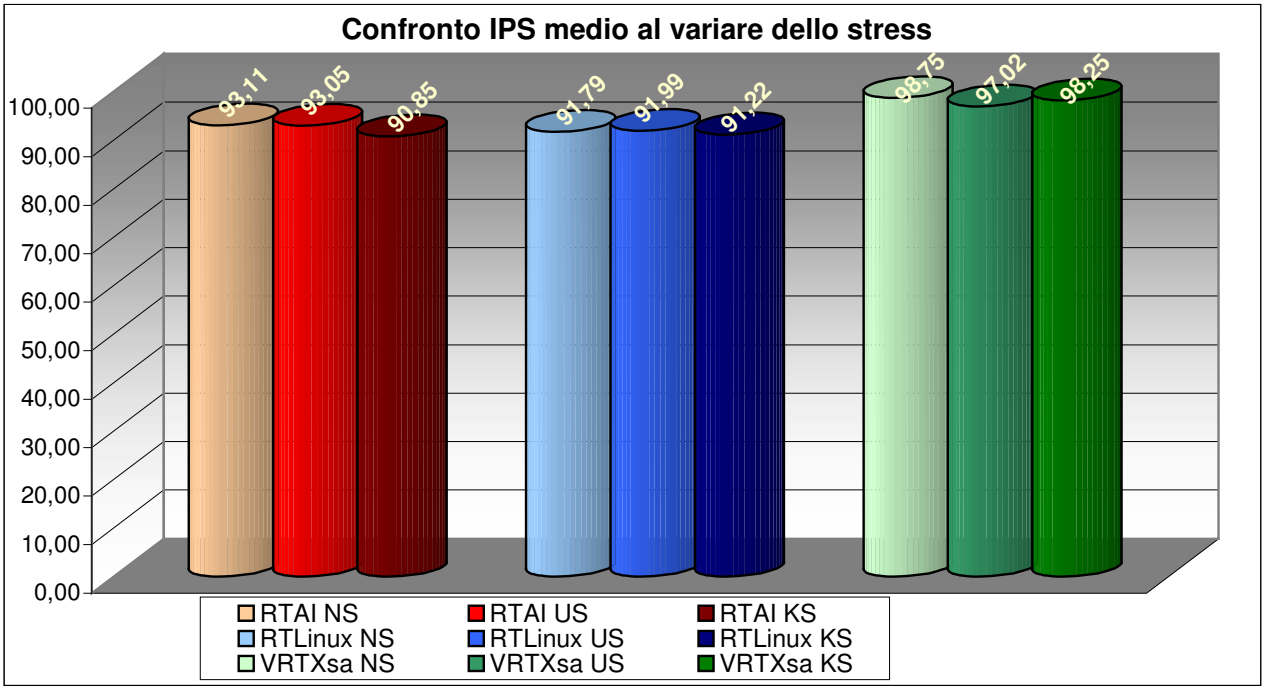

Confronti medie VRTXsa al variare dello stress
70.000
60.000
50.000
40.000
30.000
20.000
10.000
0
6.131
7.925
15.045
9.951
9.864
9.780
49.676
51.827
60.871
8.603
7.923
8.426
46.929
1.021.157
412.222
latenza fifo
efficienza fifo
IPC fifo
latenza mailboxes
IPC mailboxes
nessun carico
stress in User Space
stress in Kernel Space

2.2.2 Predictability

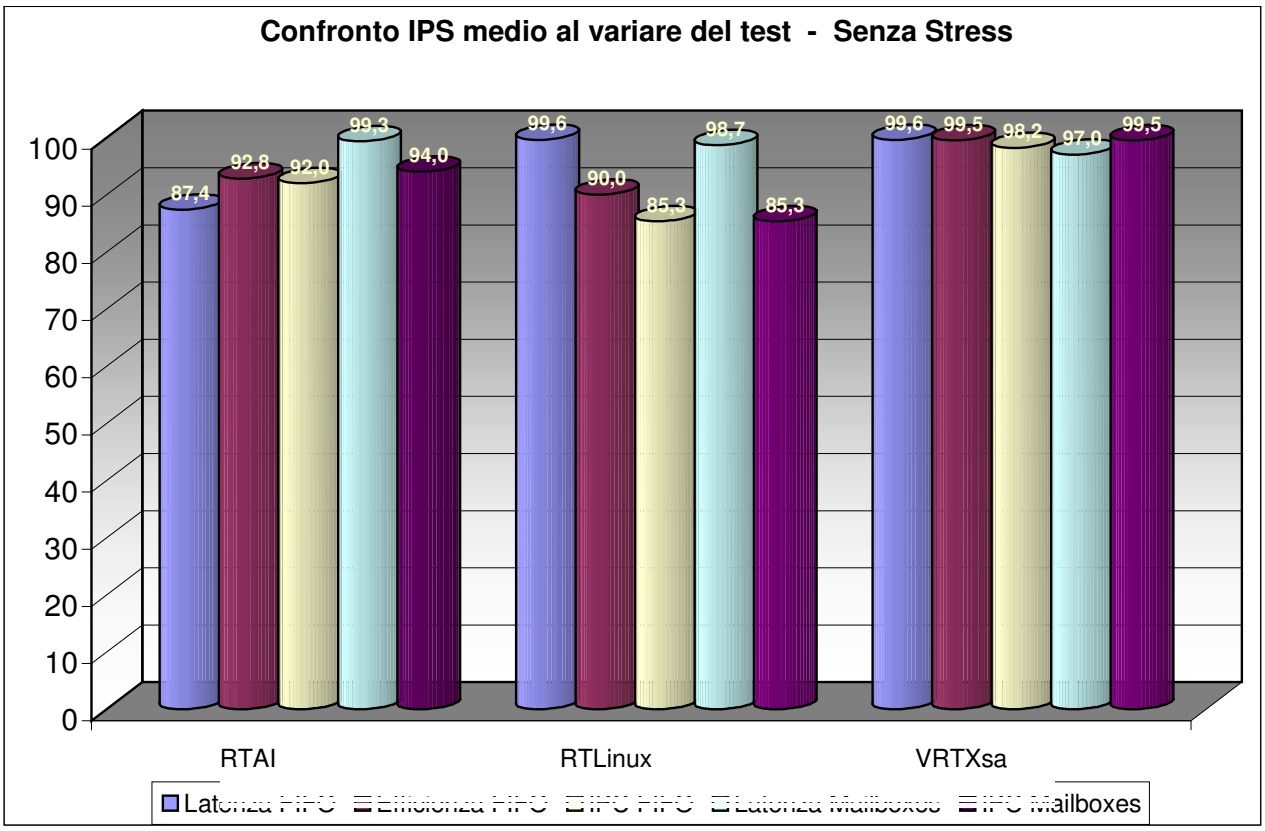
We needed to create an index in order to measure the predictability of the system.
We considered the average value of the executions of every test and increased it of a value called tolerance threshold. SPI (System's Predictability Index) is the percentage of executions which are below the threshold.

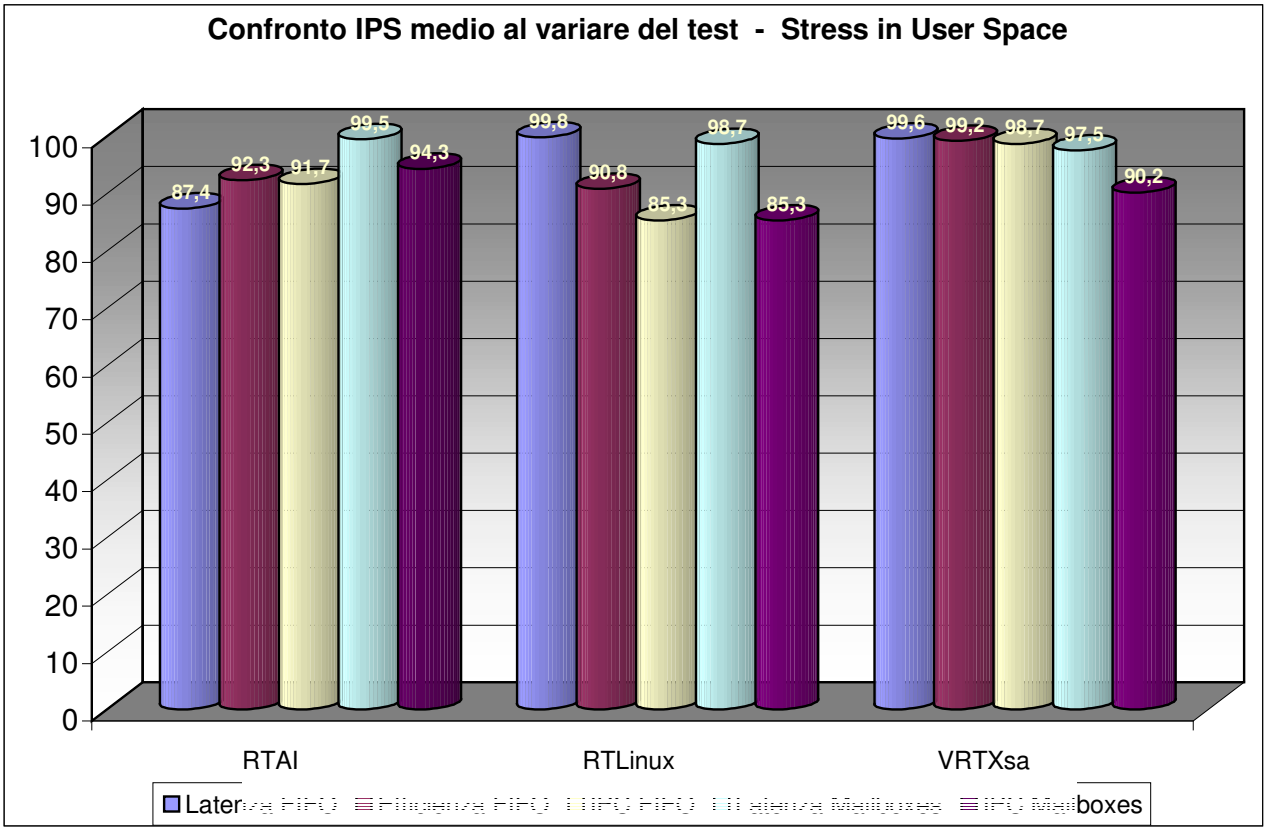


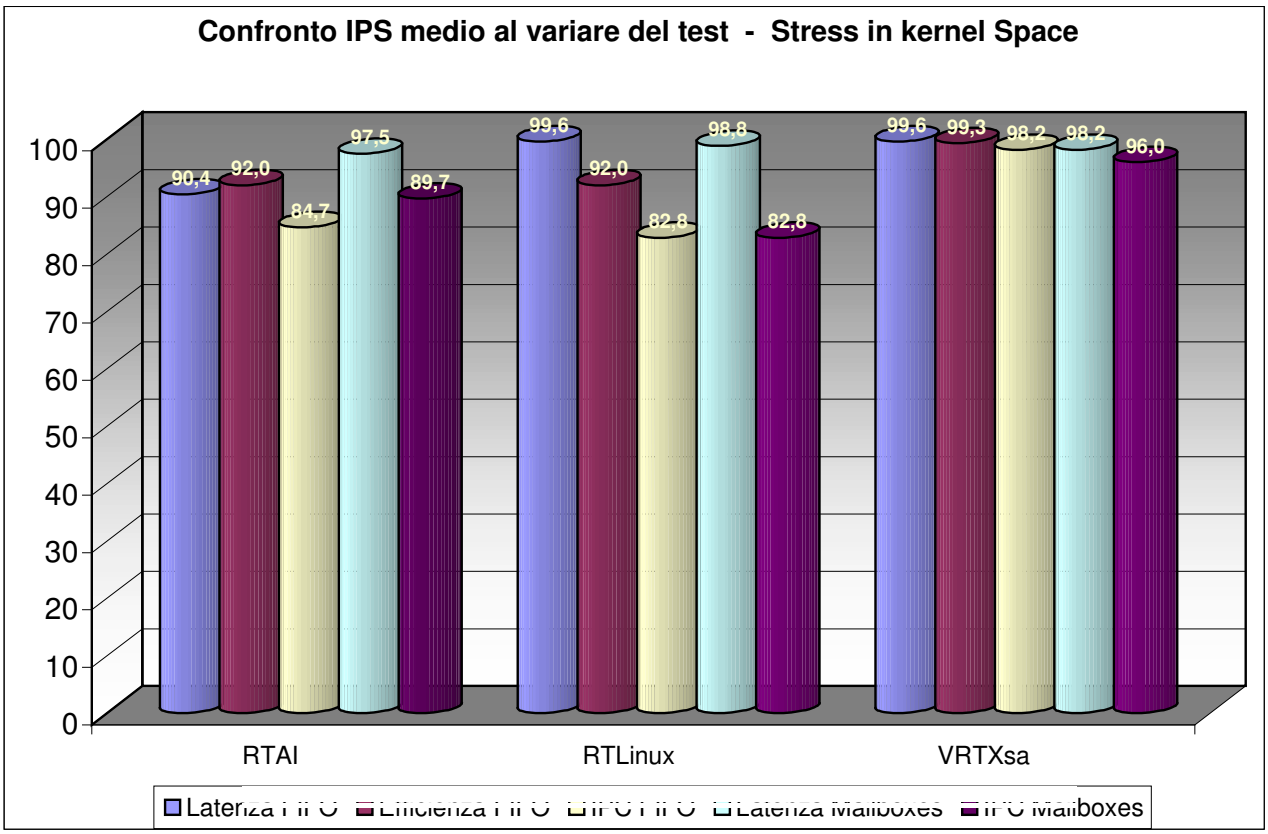
general IPS value is the average of all IPS of every test with every stress condition. We used this data to paint the General IPS comparison graph. Following graphs are about the average IPS in relation with stress condition and about IPS of every single test for every operating system.



This graph confirm the higher predictability of VRTXsa. This parameter moreover is constant if stress is increased. RTAI seems to be slightly more predictable than RTLinux. Both linux systems have a low decrease of predictability if stress is increased.


Confronto IPS medio al variare del test - Senza Stress
100
90
80
70
60
50
40
30
20
10
0
87,4
92,8
92,0
99,3
94,0
99,6
90,0
85,3
98,7
85,3
99,6
99,5
98,2
97,0
99,5
RTAI
RTLinux
VRTXsa
Latenza FIFO
Efficienza FIFO
IPC FIFO
Latenza Mailboxes
IPC Mailboxes

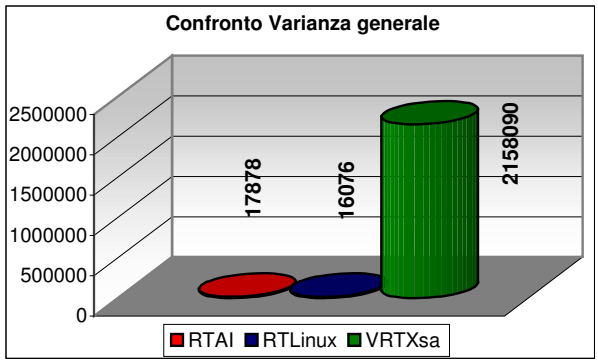

Confronto IPS medio al variare del test - Stress in User Space
100
90
80
70
60
50
40
30
20
10
0
87,4
92,3
91,7
99,5
94,3
99,8
90,8
85,3
98,7
85,3
99,6
99,2
98,7
97,5
90,2
RTAI
RTLinux
VRTXsa
Latenza FIFO
Efficienza FIFO
IPC FIFO
Latenza Mailboxes
IPC Mailboxes



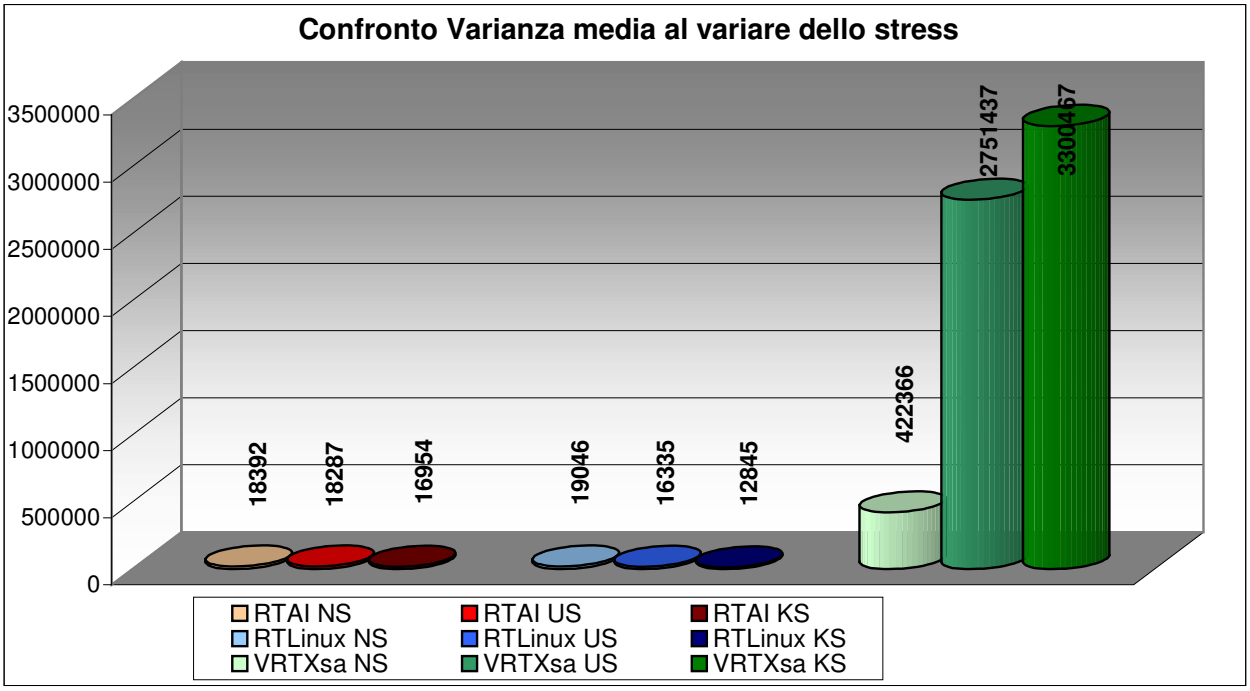
With these graphs, one for every stress condition, we're able to determine for every test which is the most predictable system. They confirm that VRTXsa is the most predictable one and that it is the less affected by stress increasing although there is a big IPC decrease in mailbox test if stress is increased.
Also linux systems generally are not very affected by stress increment, but they report a bigger IPS decrease in both ipc tests especially increasing the stress from user space to kernel space.

2.2.3 Variation

A very important parameter is the variation between the slowest and quickest execution times. So, for every test, we considered the extremis of the interval of returned values.
General Variation is the average of all variations of every test under every stress condition. We used this data to realize the graph of general variation comparison.
Following graphs are about average variation in function of stress conditions and about variation of every test for every operating system.



It's possible to see, considering even only the general variation comparison graph, how worst is VRTXsa. This is because this operating system, although generally has a less perturbed graph, reports some very high peaks which increment the variation.



In this graph we related average variation with stress increment. It's clear that Linux systems aren't affected by stress increment instead of VRTXsa.
It seems that linux based systems report a lower variation if stress is incremented. Actually what happens is that a quick system without stress is able to get very quick executions and some slower executions. If stress is increased a RTOS won't overstep the limit of the slowest executions: the result is that fastest executions will be slowed down while slowest executions will generally be constant, and the variation between slowest and quickest execution will be lower. The following schema will explain better the idea:

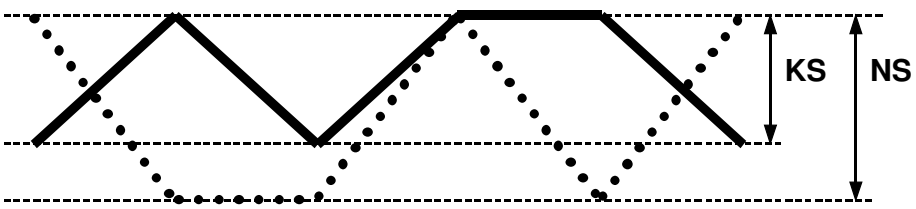
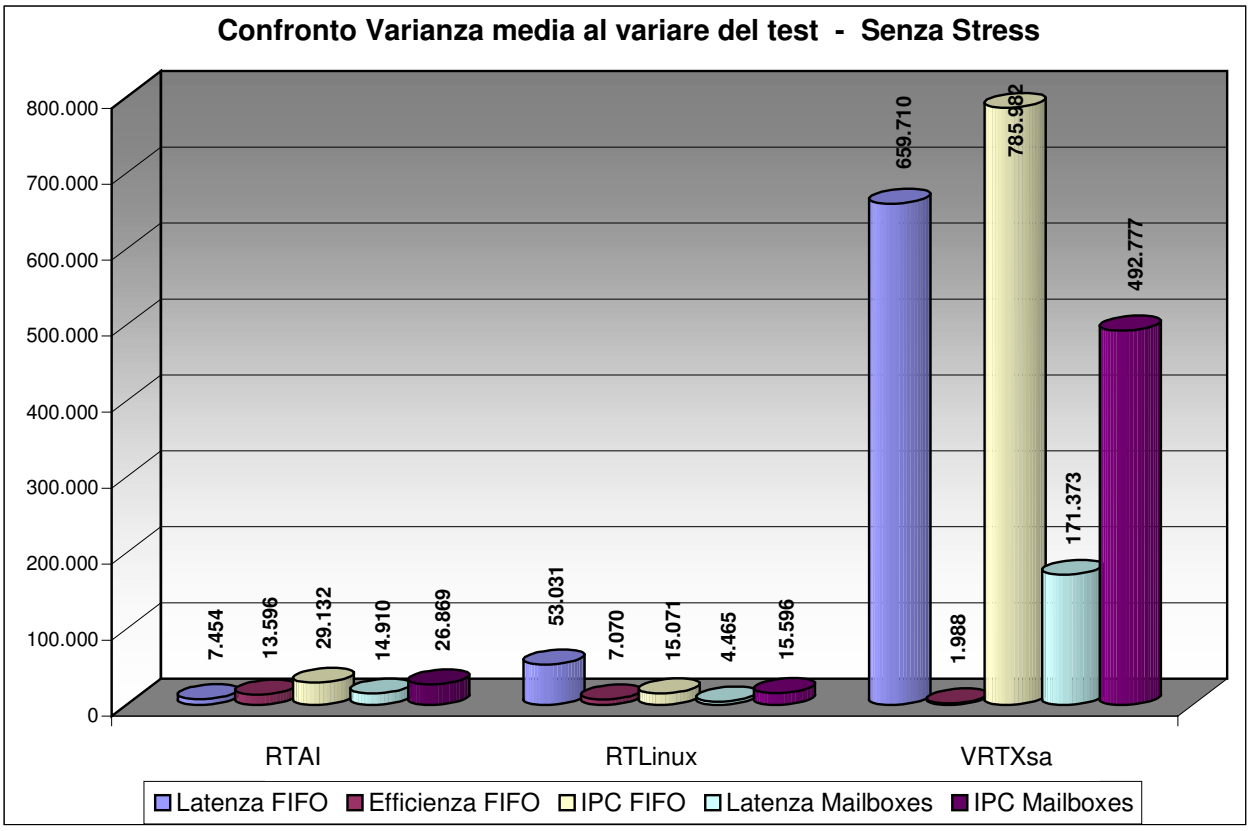
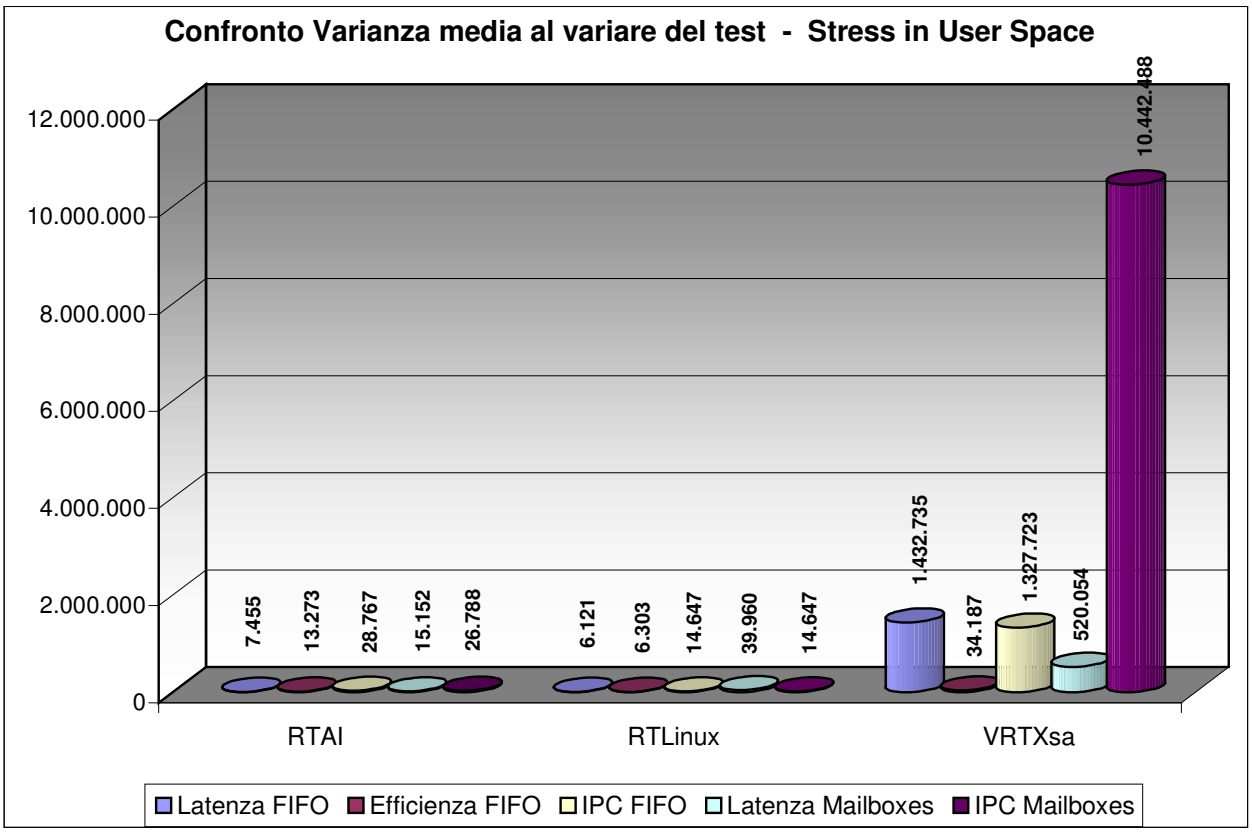
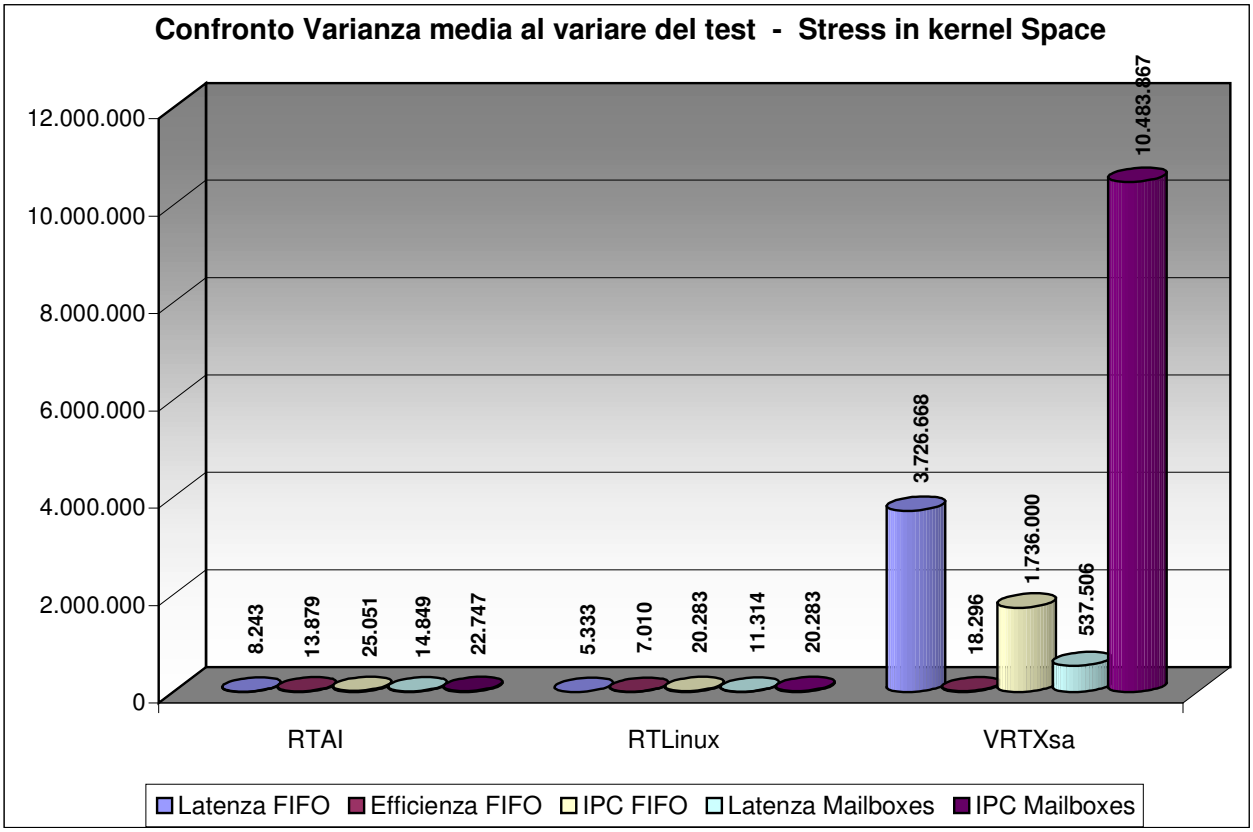


Figure 10 – Variation difference between different stress loaded test

Generally VRTXsa has less perturbed graphs, and tends to a lower variation. But it reports some peaks in some executions. Since these peaks are proportional to the stress increment the result is that variation grew up very quickly.





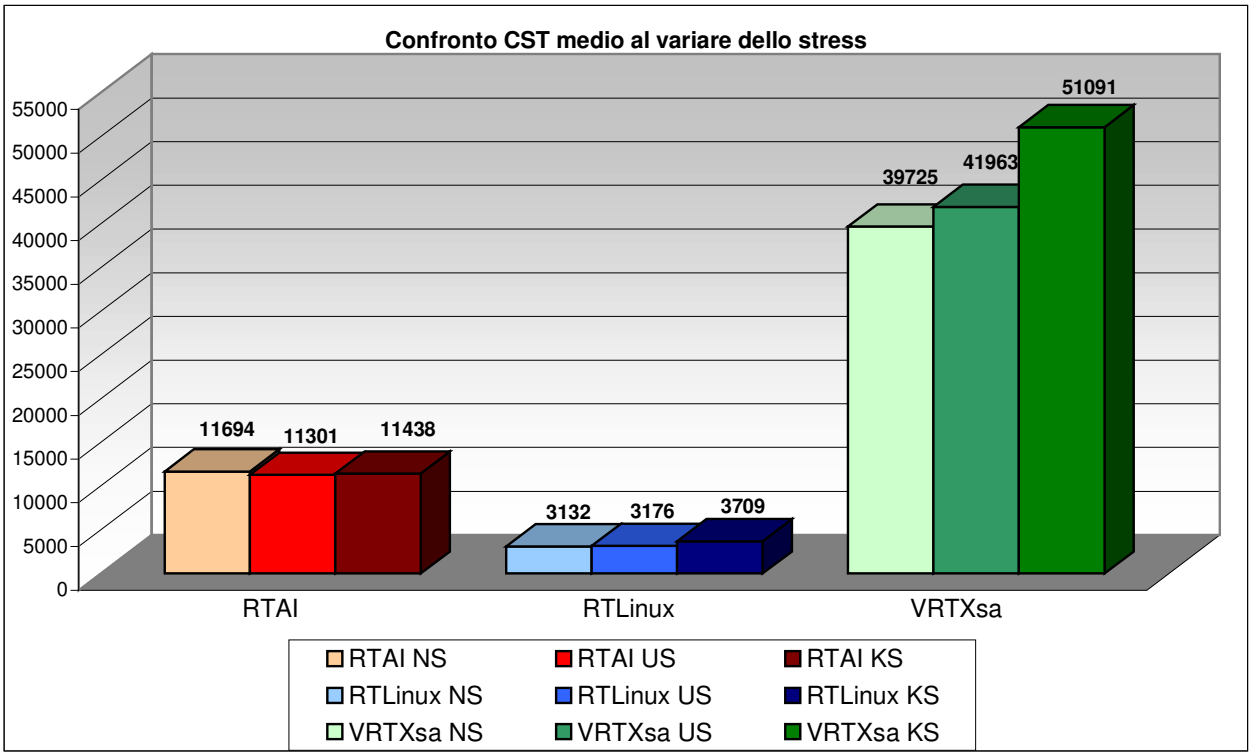


With these graphs is possible to evaluate variation of every operating system for every test.

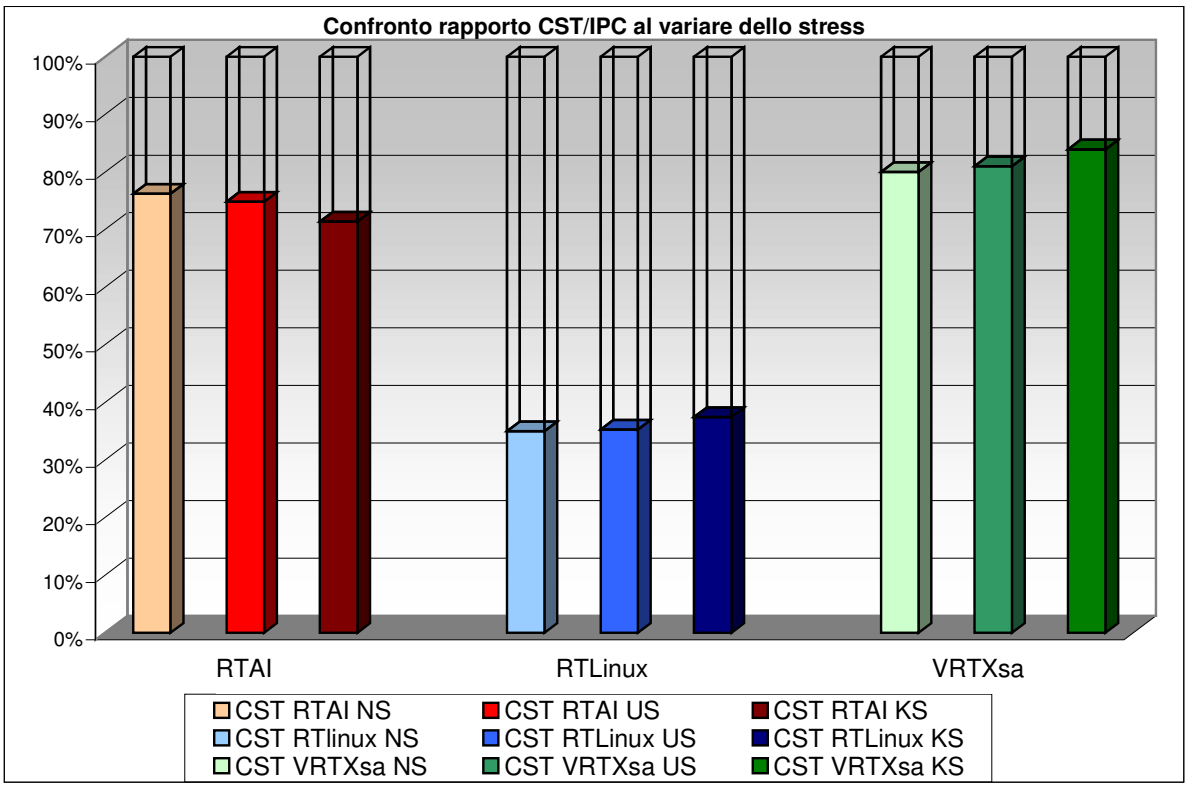
It is also possible to verify variation changing for every single test in relation with stress increment.
These graphs confirm that VRTXsa is the system with the highest variation and the most affected by stress increment. We have also to note that the first graph and the following two have a different scale.
Linux systems generally are not affected by stress increment but show a more sensible variation increment.

2.2.4 Context Switch Time

Context Switch Time (CST) is a value that measures time spent by an operating system to complete the transition of the currently executing task from executing state to ready state and inverse transition of another task ready to be executed.
This operation should take a constant time on a RTOS.
Is value has not been directly measured, but is derived from reprocessing of data already collected using fifo and fifoCS tests.
If we detract values obtained from fifo.c from measurement returned from fifoCS.c we'll get an indicative measurement of CST.

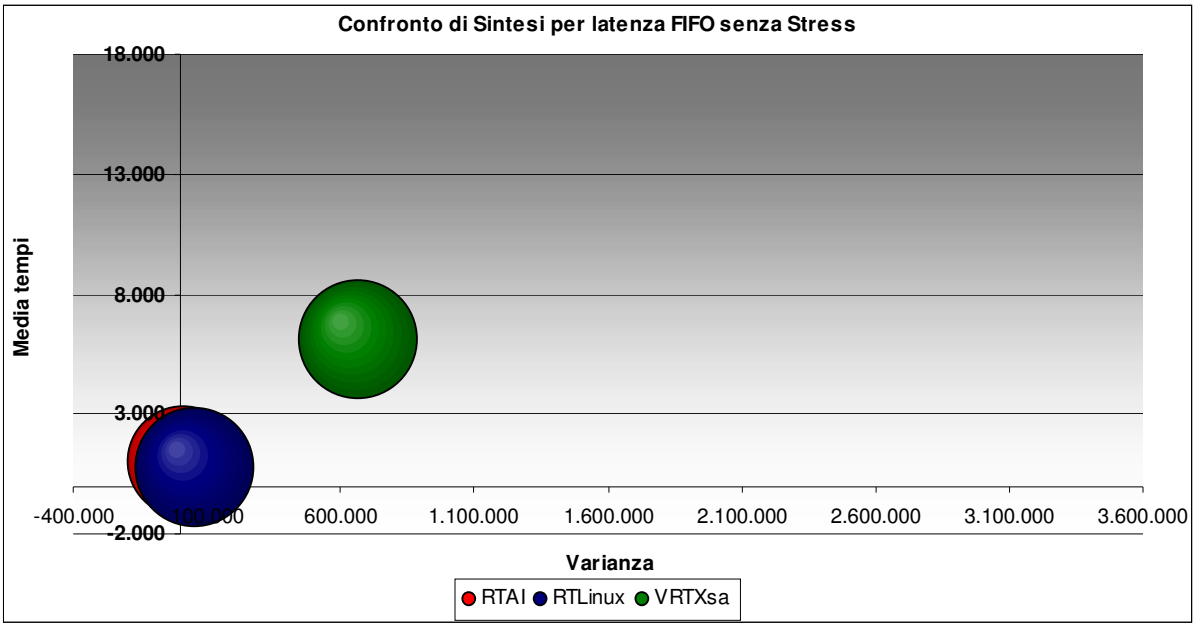


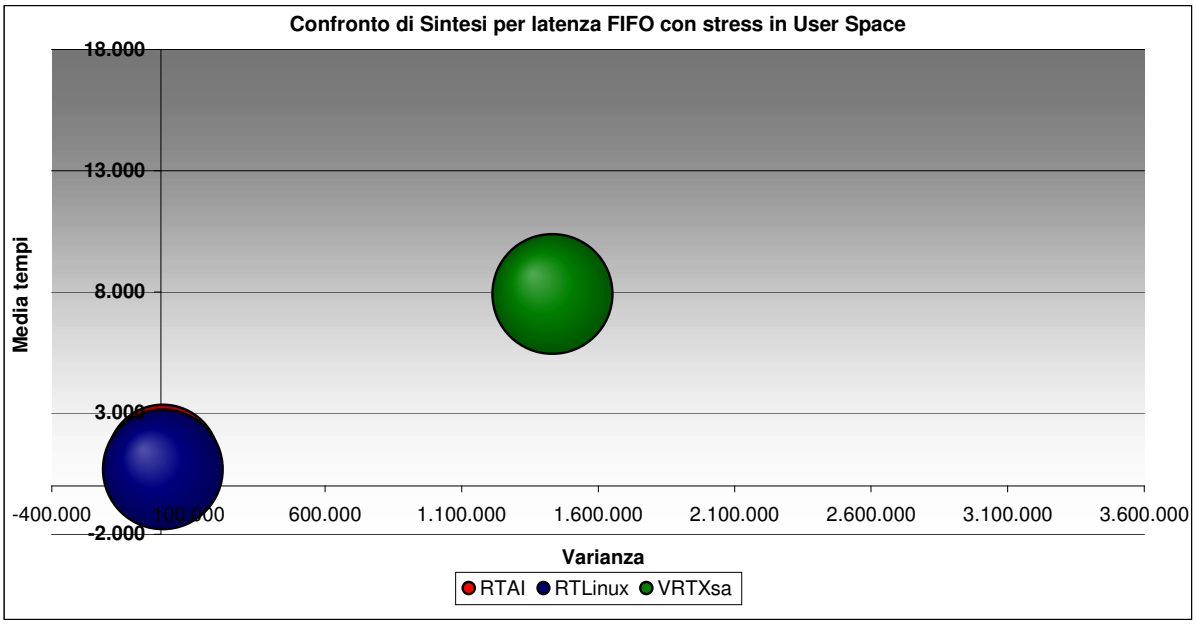
In this graph is clear that CST times are nearly constant in linux systems but they increase clearly in VRTXsa when stress is increased.
The next graph instead reports in percentage the impact of CST and of overhead IPC intratask on the value returned from fifoCS.c.
We don't have to read from the following graph that RTAI is a system that takes less time to execute the context switch if under stress. We can understand instead that IPC intratask overhead graves proportionally more on the total time than the CST in the case of kernel space stress.
It's mode interesting to consider that, proportionally on the time reported by fifoCS.c, VRTXsa takes more time to execute context switch than linux based systems. Actually RTLinux seems to be the system which in proportion takes less time to switch the context: the time spent by RTLinux to switch from a task to another one is about the 35% of the time reported by fifoCS.c without differences depending from stress. The 65% of the time is spent to execute the IPC intratask. This data confirms what already seen in the efficiency comparison graphs where RTLinux was slower than RTAI undepending by the stress.
VRTXsa takes the 80% of the time to switch the context and finally RTAI takes about the 75% of time reported by fifoCS.c.
Differences related to stress increments are unimportant except for the case of RTAI in kernel space: in this test IPC intratask overhead graves more.

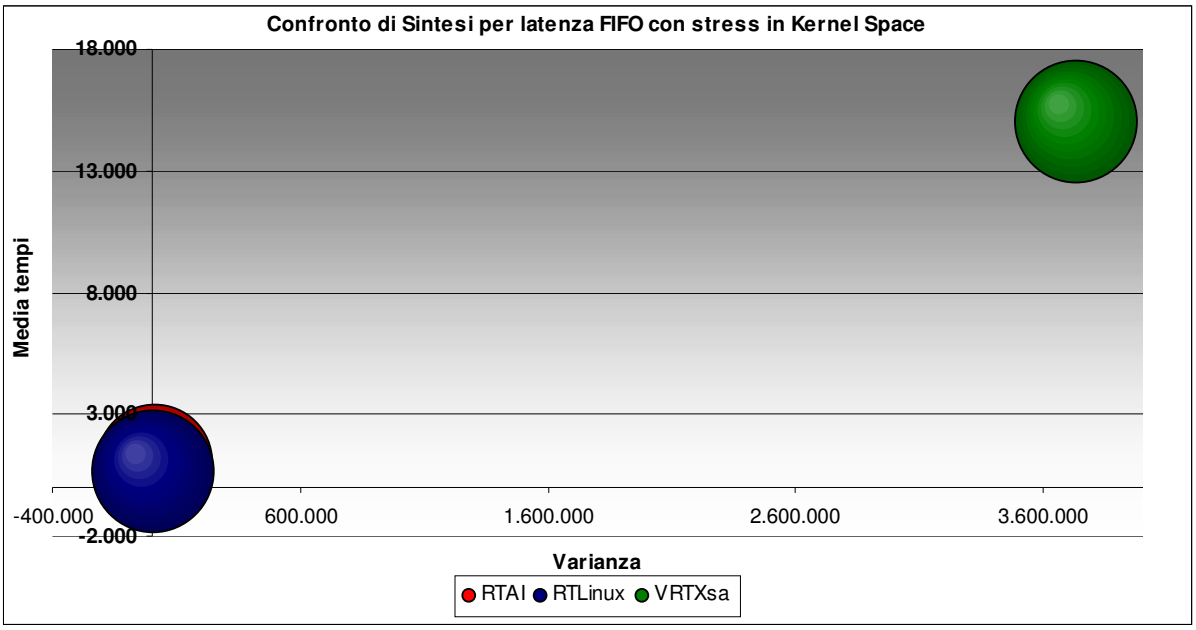

Confronto rapporto CST/IPC al variare dello stress
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
RTAI
RTLinux
VRTXsa
CST RTAI NS
CST RTAI US
CST RTAI KS
CST RTlinux NS
CST RTLinux US
CST RTLinux KS
CST VRTXsa NS
CST VRTXsa US
CST VRTXsa KS

2.3 Final Comparisons

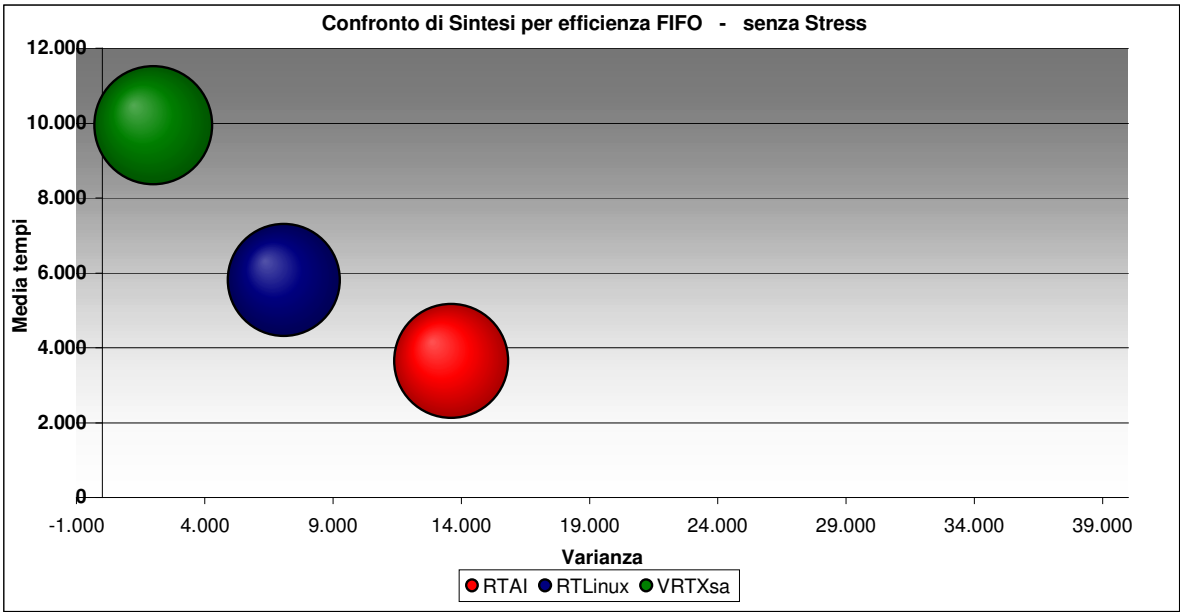
Efficiency, Variation and Predictability are the three main parameters we detected to evaluate performances of a RTOS. Therefore we prepared some summary graphs just to put together these values.
Variation is on x axis, efficiency on y axis (as execution average time). Each operating system is represented by a sphere which radius is its predictability.
The optimum operating system is then the one that minimizes variation and execution times and maximizes the predictability. It will be represented by a sphere of maximum radius (100) situated just in the origin of axes.

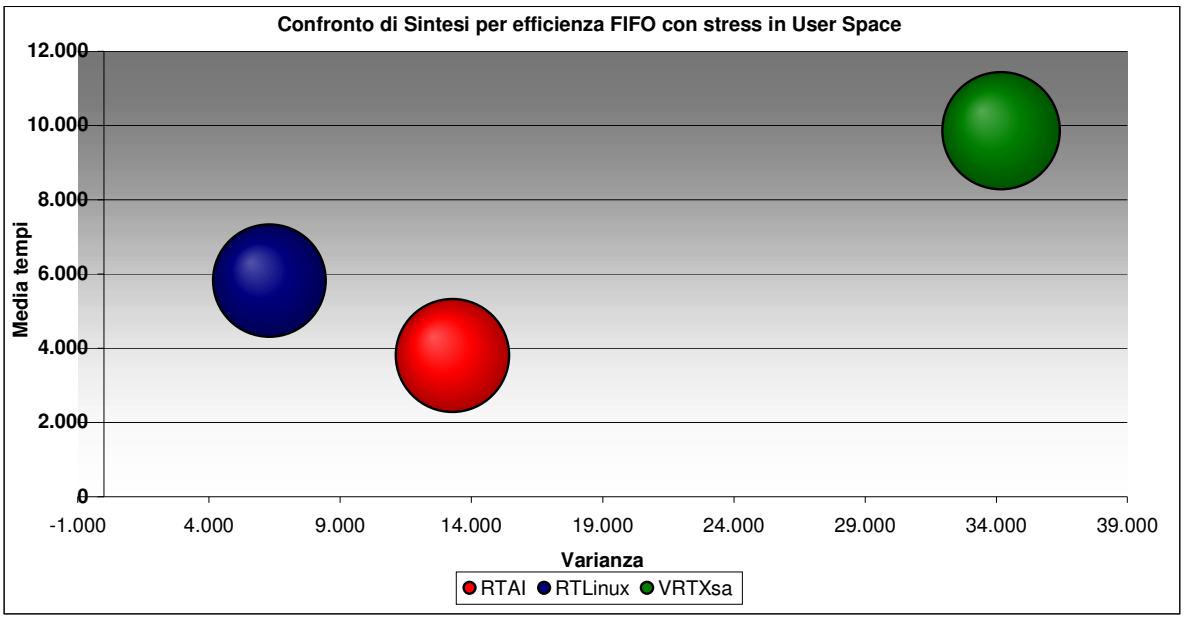


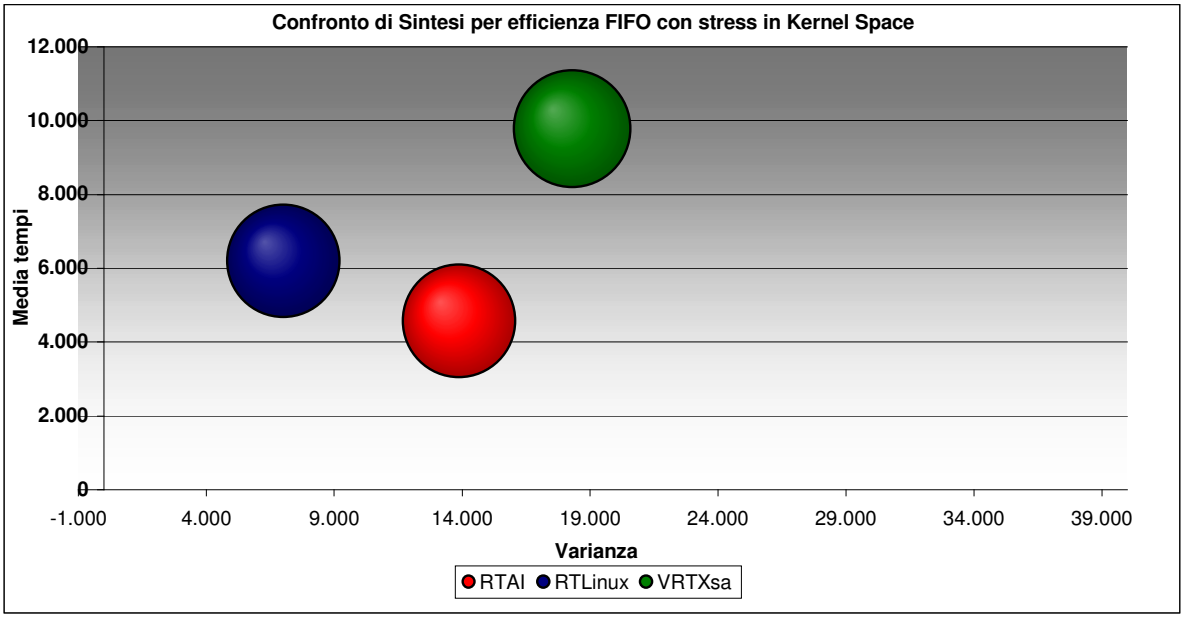




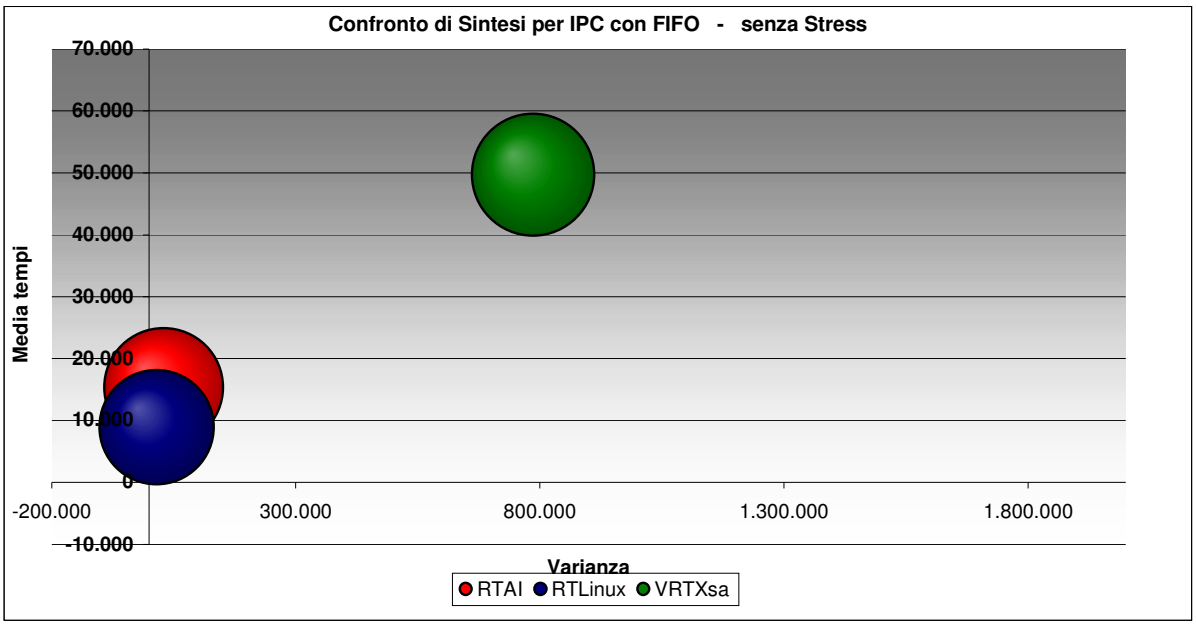
It's evident in these first three graphs the VRTXsa worsening if stress is increased, while performances of the other systems are nearly constant.
About predictability all the systems are comparable.

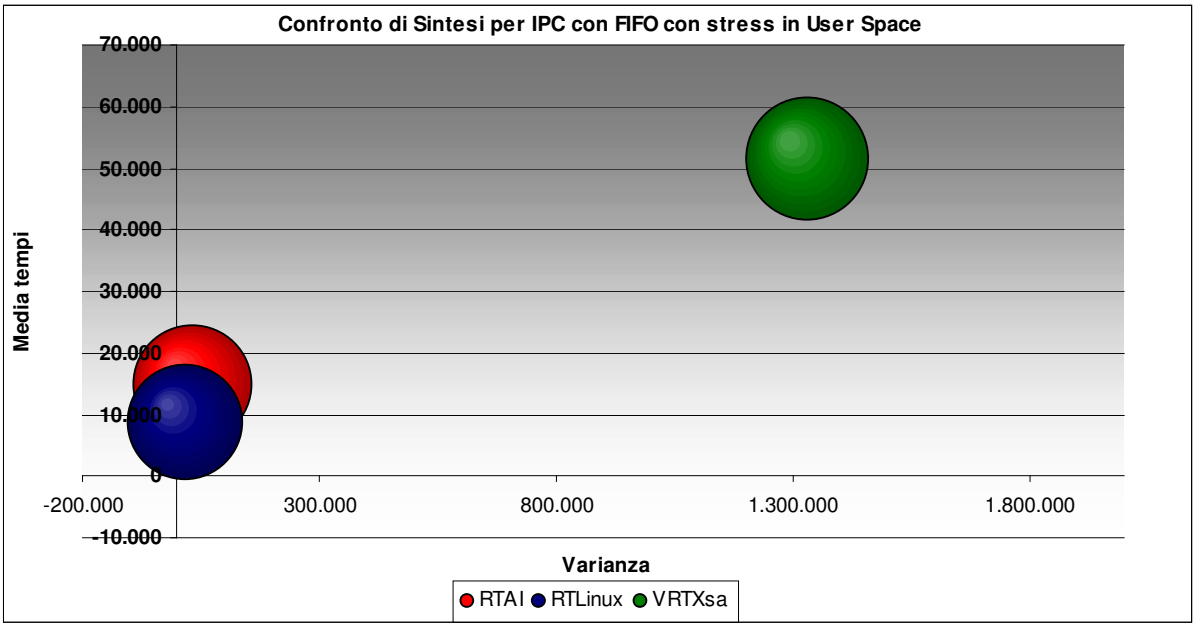


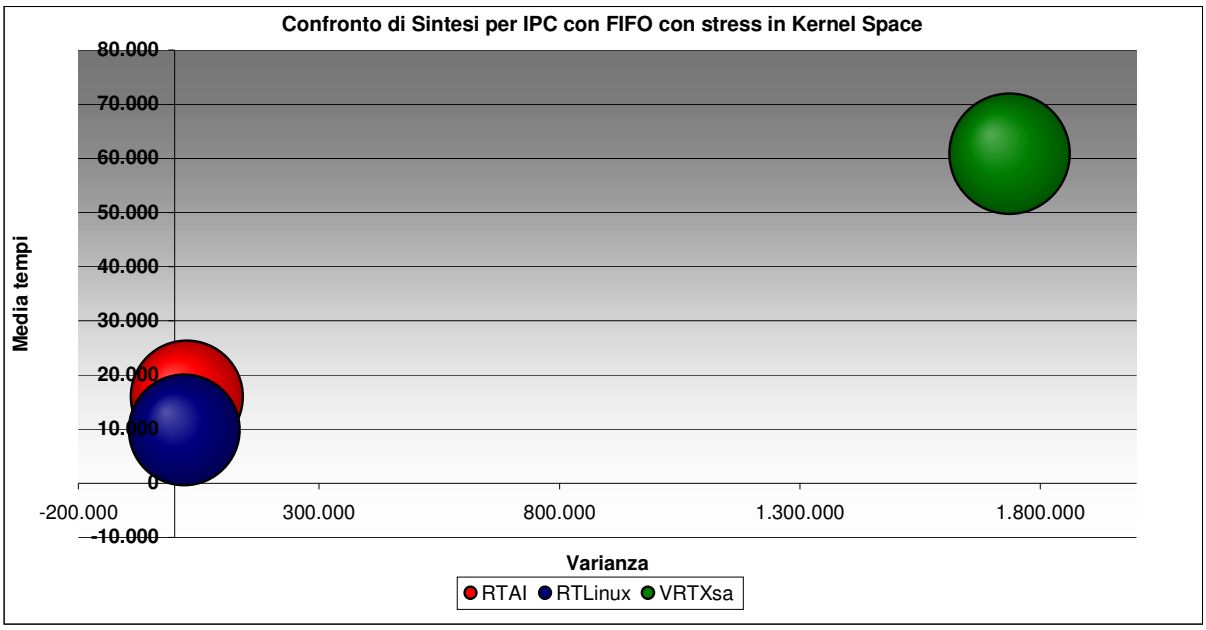




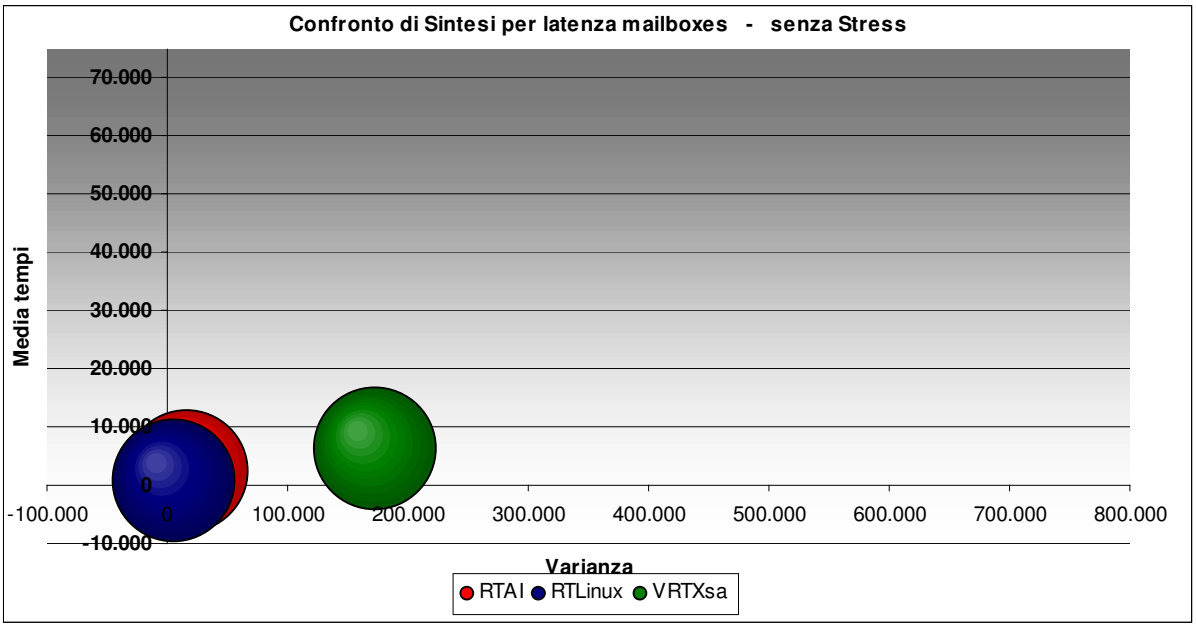
Also in this case linux based system report constant variation, efficiency and predictability.
RTLinux is the best system about variation. RTAI instead is the best about efficiency.
All the systems are comparable about predictability. Again VRTXsa is very far from the origin.

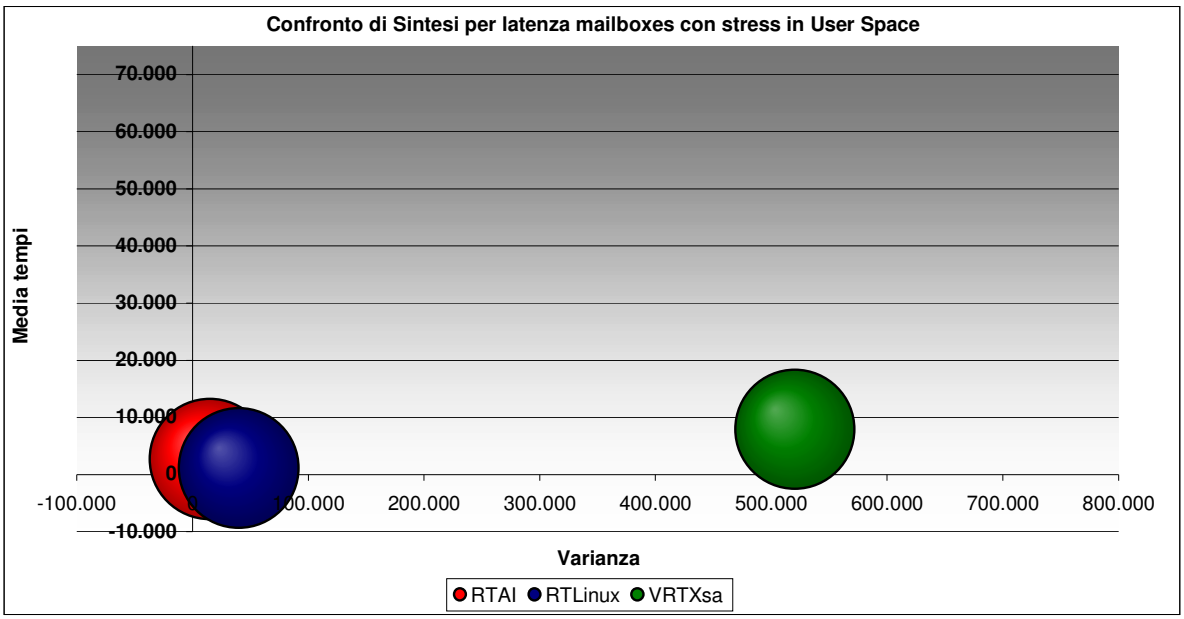

Confronto di Sintesi per IPC con FIFO - senza Stress
70.000
60.000
50.000
40.000
30.000
20.000
10.000
0
-10.000
Media tempi
-200.000
300.000
800.000
1.300.000
1.800.000
Varianza
RTAI
RTLinux
VRTXsa

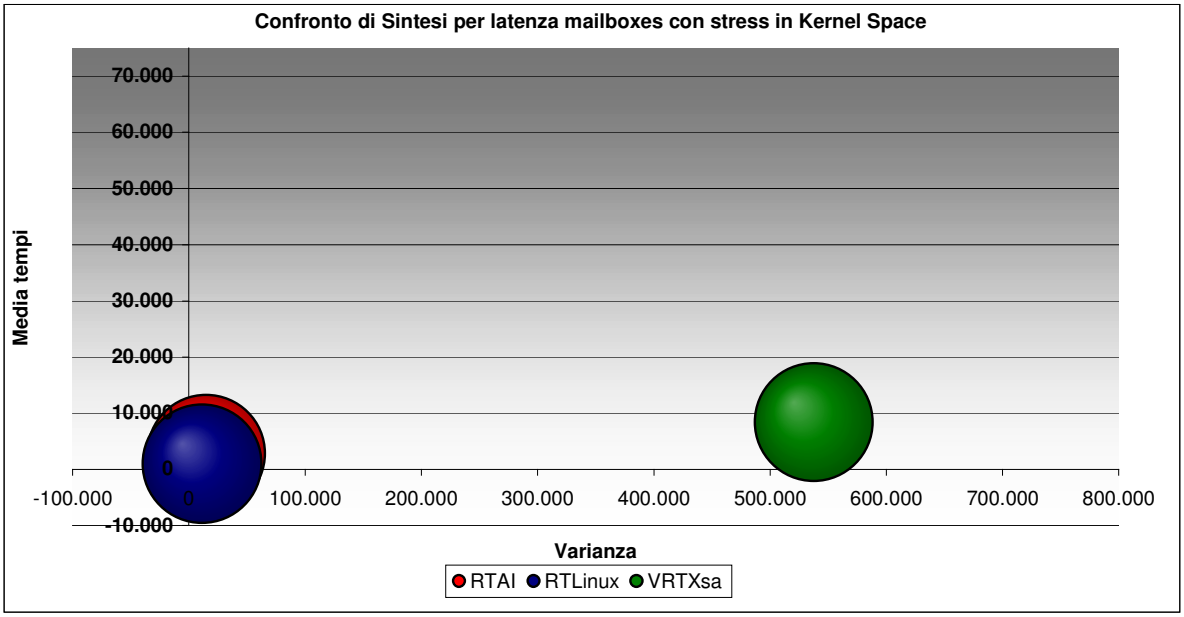

Confronto di Sintesi per IPC con FIFO con stress in User Space
70.000
60.000
50.000
40.000
30.000
20.000
10.000
0
-10.000
Media tempi
-200.000
300.000
800.000
1.300.000
1.800.000
Varianza
RTAI
RTLinux
VRTXsa



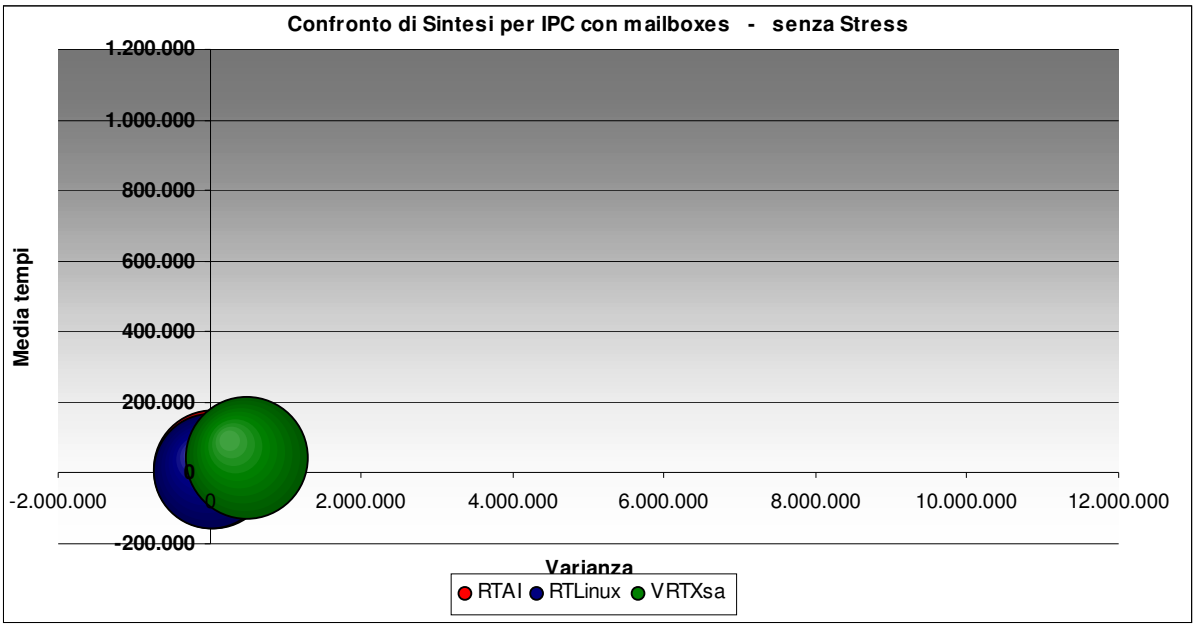
First of all we have to consider the different scale of these three graphs. VRTXsa is very far from the origin and it still moves away increasing the stress. In these, and the following graphs linux systems are always situated very close to the origin instead of VRTXsa. About predictability all systems are comparable.

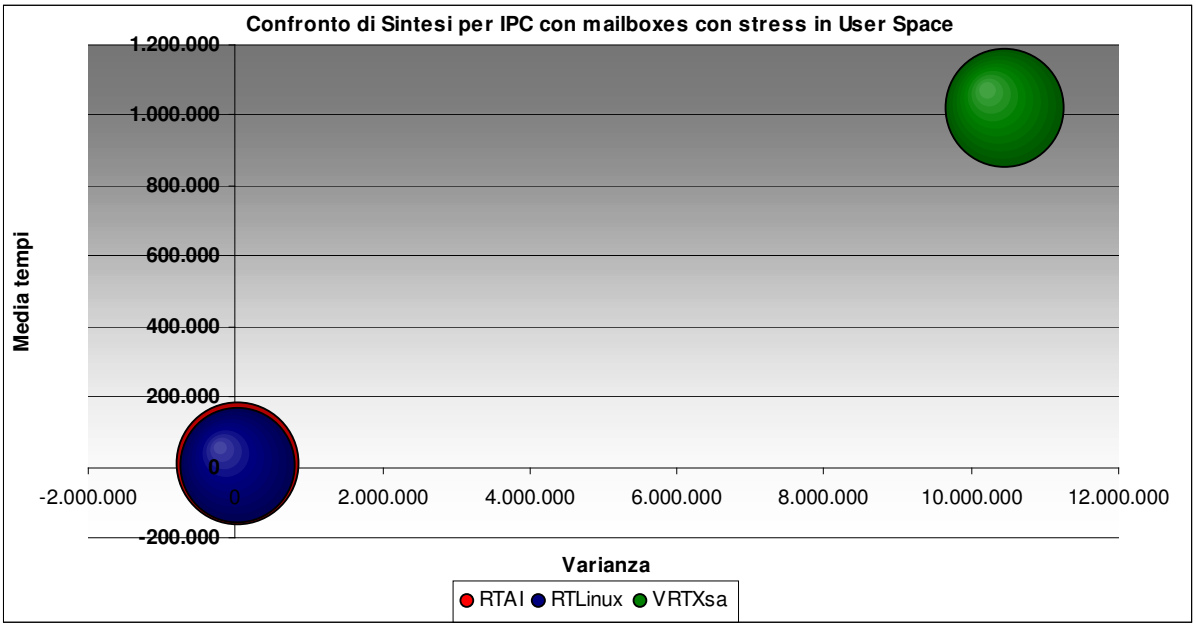


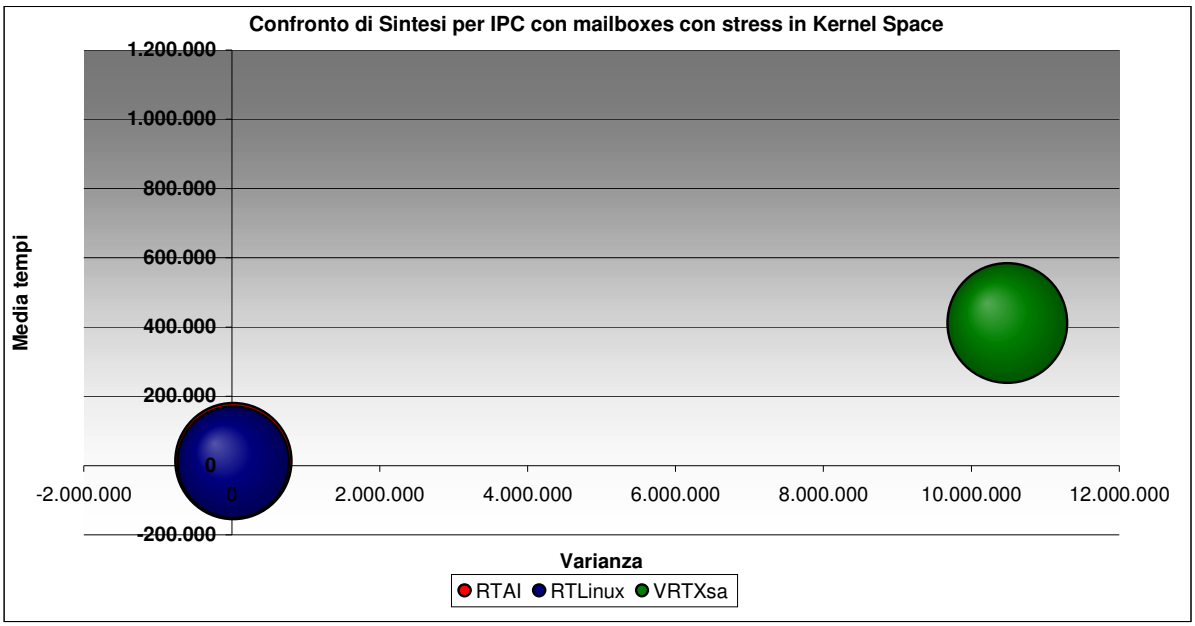




About efficiency and predictability all the systems are comparable. VRTXsa is a little bit slower but the main fact is that it reports a variation worsening if stress is increased.
Linux systems' spheres are very close to the origin.


Confronto di Sintesi per IPC con mailboxes - senza Stress
1.200.000
1.000.000
800.000
600.000
400.000
200.000
0
-200.000
Media tempi
-2.000.000
0
2.000.000
4.000.000
6.000.000
8.000.000
10.000.000
12.000.000
Varianza
RTAI
RTLinux
VRTXsa


Confronto di Sintesi per IPC con mailboxes con stress in User Space
1.200.000
1.000.000
800.000
600.000
400.000
200.000
0
-200.000
Media tempi
-2.000.000
0
2.000.000
4.000.000
6.000.000
8.000.000
10.000.000
12.000.000
Varianza
RTAI
RTLinux
VRTXsa



The scale of these graphs may let think that the systems are comparable at least without stress.
Instead about variation and efficiency VRTXsa has worst performances. RTLinux and RTAI are aligned about variation and efficiency but RTAI is better about predictability.
Once again is evident the VRTXsa performance worsening related to the stress increasing.

3. Conclusions

It seems generally clear that linux systems are more performing considering efficiency while VRTXsa has the less perturbed graph.
It seem evident that linux systems are the best choice considering the following facts:

- maximum efficiency values are always lower than VRTXsa minimum values
- linux based systems peaks are more frequent but more modest than VRTXsa ones
- VRTXsa in ipc tests has the advantage of managing a quarter the data linux manages (4 bytes against 16)

Considering all these facts and what analysed previously, it seems that linux based systems are the best solution and that such systems can be a valid and more efficient alternative to VRTXsa. Therefore would be surely advisable a migration from VRTXsa to a linux based real time system.