Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 849 – 858
ISSN 1896-7094
© 2007 PIPS

Task jitter measurement under RTLinux operating system

Pavel Moryc, Jindřich Černohorský

Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Measurement and Control
Centre for Applied Cybernetics
Ostrava, Czech Republic
pavel.moryc@mittalsteel.com
jindrich.cernohorsky@vsb.cz

Abstract. This paper deals with real-time task jitter measurement under RTLinux operating system. In the first part, it describes methods and tools developed to measure jitter in the RTLinux environment. In the second part, it is focused on discussion of results, obtained on PC hardware, and their interpretation.

1. Introduction: Applicability of Linux in Real-Time Systems

In a real-time system there is a conflict between periodic and aperiodic tasks. Aperiodicity naturally stems from noise, disturbances, delays, and all other unpredictable phenomena in the real world. A real-time operating system should not enforce strict rules that nature cannot meet, but rather provide resources that help to smooth the conflicts. Basic approaches are preemptivity, buffers, and priority rules. However, as a result of preemptivity, a high-priority task requiring a resource may be blocked to wait for medium-priority tasks that do not hold this resource. This problem is called priority inversion.

Linux is neither intended, nor designed to support real-time tasks. It is a general-purpose operating system, implementing full range of API functions covered by the POSIX-1003.1 specification. A kernel providing such large scope of API services cannot meet demands of preemptivity and low latency, required in most technology control systems.

RTLinux kernel implements a Hardware Abstraction Layer inserted between hardware and the Linux kernel. Essentially, it creates a virtual machine, that controls the Linux kernel timer interrupt. The RTLinux can switch between the Linux kernel and other tasks [1, 2], thus making possible to solve conflicts between real-time tasks and the Linux kernel. The Hardware Abstraction Layer (HAL) controlling the system is realized in the Linux kernel space. The system as a whole is simple and fast, but barriers between real-time task and the non-real-time Linux kernel are thin, and as a result of that, the real-time part may easily get out of stability margins required.

Jitter is a variable deviation from ideal timing event. Scheduling jitter is the delay between the time when task shall be started, and the time when the task is being started. Similarly, interrupt latency is a delay between the interrupt is triggered and the time when the Interrupt Service Routine is being started. The interrupt latency varies, and therefore, it is a jitter. Jitters result from physical phenomena in hardware (noise), from concurrent task processing (realized either in hardware or in software), and from passing the code through different branches (each conditional instruction is a potential jitter source). Kernel latency is not stable, but composed from various phenomena, most of them (if not all) showing jitters.

RT-Linux is designed as a module to the Linux kernel, and therefore, it could be reasonably supposed, the RTLinux kernel can suffer from jitters inherited to the Linux kernel. Hence, thoughtful testing is of prime importance.

2. Existing measurement methods

Throughout the time, many jitter measuring methods were developed and published [3, 4]. Periodic task is characterized by its starting time of execution and by the length of execution, while aperiodic task is characterized by its latency. Interrupt latency is defined as the time from generating the interrupt request to (the start of) its service routine execution.

Both theory and experience requires, that a system shall be tested under load. In a technology control system, the load is caused by the specific application requirements. But, whenever results with more general validity are needed, a more general load shall be applied. As a model load, heavy network or disk load is often used. Proctor [3] has performed evaluation of these load types influence on RTLinux operation. Unfortunately, [3] does not include specifications of used hardware.

Published methods of RTLinux evaluation are either concentrated on common techniques of jitter and latency measurement [3] or focused on measuring latencies of basic RTLinux resources [4].

However, above mentioned papers present specific approaches and methods rather than general application studies. The applicability of obtained results to a different design seems to be uncertain.

3. Designed measurement methods

This work presents an approach based on a generalized application program. It is intended as a measuring method tightly connected with application study. Therefore, it can be reasonably assumed, that results will be useful for general engineering practice.

3.1 Generalized data acquisition program

RTLinux in tight connection with Linux is intended for use in systems, that allow both real-time and non-real-time tasks to coexist, and it is typically applied as an interface between dedicated real-time and non-real-time IT levels. As a representative case of this class, a generalized data acquisition program has been chosen, supplied with diagnostic timestamp outputs.

The application, named RT-golem, contains and integrates resources, which make possible to apply a defined workload to the system, as well as to measure how the load task is executed on the system. It implements both periodic and aperiodic tasks. The RT-golem is a part of an overall architecture presented in Figure 1. The architecture contains and integrates resources, which make possible to apply a defined workload to the system, as well as to measure how the load task is executed on the system. The RT-golem includes

- periodic task, which is controlled by RTLinux scheduler,
- aperiodic task (the Interrupt Service Routine, which is installed instead the default RTLinux ISR routine).

During initial tests performed with the RT-golem it was observed, that excessive IRQ requests can disrupt system operation. For that reason, the RT-golem was strengthened with overload protection, so it can sustain arbitrary input IRQ rates. Based on the test results, a saturation method of measuring interrupt latency was designed, as shown in Figure 2. Interrupts are triggered by a periodic signal supplied from external generator. The incoming interrupt rate is boosted, till the time between two successive ISR routine starts decreases. When the time stops decreasing, the IRQ rate reaches its saturation point. The minimum time between two successive ISR starts equals to the interrupt latency. It consists of latency times caused both by hardware and software resources.

Since the saturation imposes the maximum IRQ rate load on the measured system it is capable to accept, the method is expected to provide comparable results across various hardware platforms.

RT-golem application has been further modified, so it could be loaded more than once. This creates a possibility to load the system by more periodic tasks, each with different priority, period, time of execution and diagnostic timestamps output. It has evolved to a configurable and flexible simulation tool.

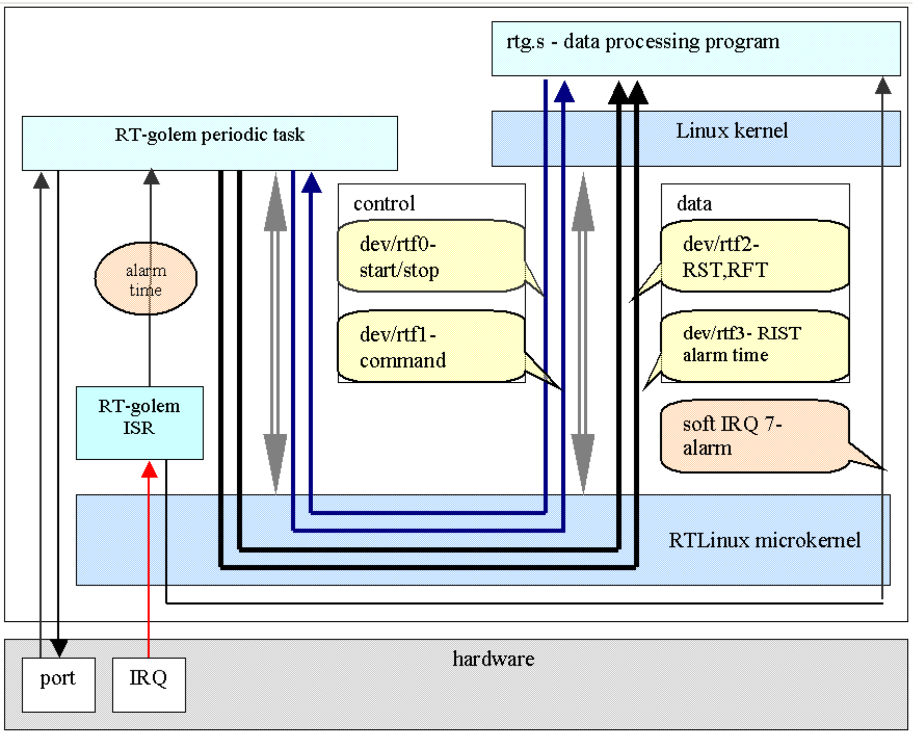
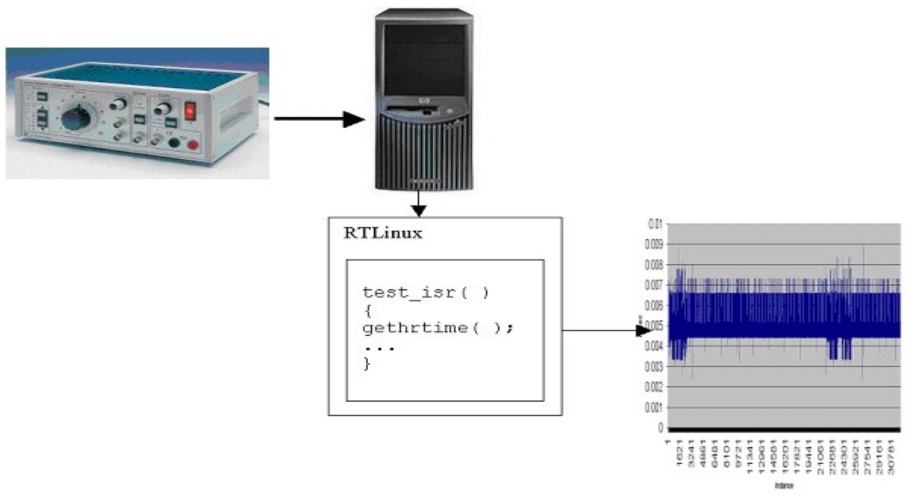


Fig. 1. RT-golem operation



```
test_isr( )
{
gethrtime( );
...
}
```

Fig. 2 . ISR latency saturation method

3.2 Advanced measurement tool: RT-golem

Analysing the measurement results and RTLinux resources, it has been recognized, that a measurement tool, which encompasses the whole range of typical RTLinux resources and provides a deeper insight, is needed. Based on this analysis, the following important RTLinux characteristics have been identified:

- precision of the scheduler (measured as task starting time jitter),
- interrupt latency time,
- execution time of typically used API services,
- pipe write and read operations,
- shared memory write and read operations,
- thread switching time
- I/O port read and write access time.

The I/O access is also included, because it characterizes hardware, and presents the basic method of communicating with both sensors and actuators.

The generalized application was substantially redesigned to form an advanced measurement tool. The advanced version of RT-golem consists of a periodic task, and an interrupt service routine. The periodic task includes two threads. It is possible to set priority and period of both threads, as well as to disable one or more parts of the task. This way, it is possible to balance the workload, that the RT-golem imposes on the system.

4. Experimental setup

A set of comparison measurements has been performed. In particular, the effects of different load on different test systems have been measured, as follows:

- no load,
- load with copying files
  (while [true]; do cp /bin/bash ${f}; done),
  /bin/bash is ca. 70kB in length,
- load of 15 RT-golem 5.1 tasks

on two test systems,

- PC Dell GX 280,
- PC no name.

The source code of the RTLinux scheduler contains a comment [5] recommending that this scheduler should not be used for more than 10 tasks. For verification of this recommendation, an experiment was designed, where the system is heavily loaded by fifteen RT-golem tasks, and jitters of RT-golem test task are measured. The fifteen tasks have been configured as maximum acceptable load for the system, that is, the highest load, at which the Linux kernel yet does not start reporting lost timer interrupts.

PC Dell is a workstation designed for graphical applications, while PC noname is a low-cost, low-end personal computer. Linux kernel has been configured to use only 64 MB of RAM memory. Test system configurations are presented in Table 1.

Table 1 . Test system details

PC DELL GX 280

CPU	Intel P4 3.0 GHz, 1 MB L2 cache
RAM	1024 MB
HDD	SAMSUNG SV0842D, SATA, 75GB
	WDC WD800JD-75JNC0, 8 GB, ATA-66

PC no name

CPU	Intel P4 2.4GHz, 32 K of L1 cache
mainboard	MSI 865 PE Neo2-P
RAM	256 MB
HDD	Seagate Barracuda ST380011A 80 GB ATA-100
	Maxtor WDC WD100EB-00BHF0 10 GB ATA-66

5. Experimental results

Because of limited space, only a handful of results can be presented. The first series of graphs, presented in Figures 3 through 6, show the task instance starting (or finishing) time, while the second series of graphs (presented in Figures 7 and 8) shows the statistical data. The task instance starting time is calculated from the previous task instance starting time. This means, the starting time delay impacts two adjacent values. First, the difference between the correct and delayed instance is longer, which causes the spike up on the graph, and then, the difference between the delayed and next correct instance is shorter, which causes the spike down. If both spikes are symmetrical, the second value is okay. Finishing time is calculated from task instance starting time.

Spikes on the relative starting time graphs below oscillate around 1 msec, because they show scheduling jitter, that means, a difference of the actual relative starting time from the nominal value, which is 1msec.

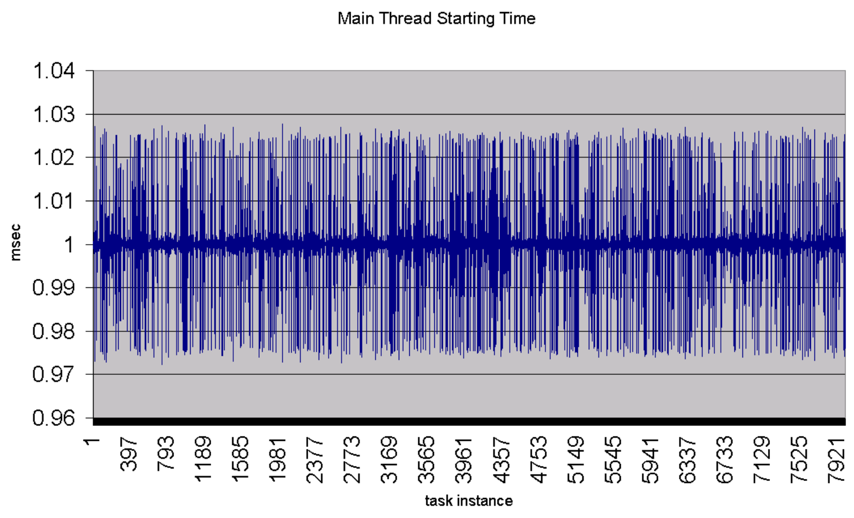


Fig. 3. Periodic task starting time, PC Dell, loaded with copying files

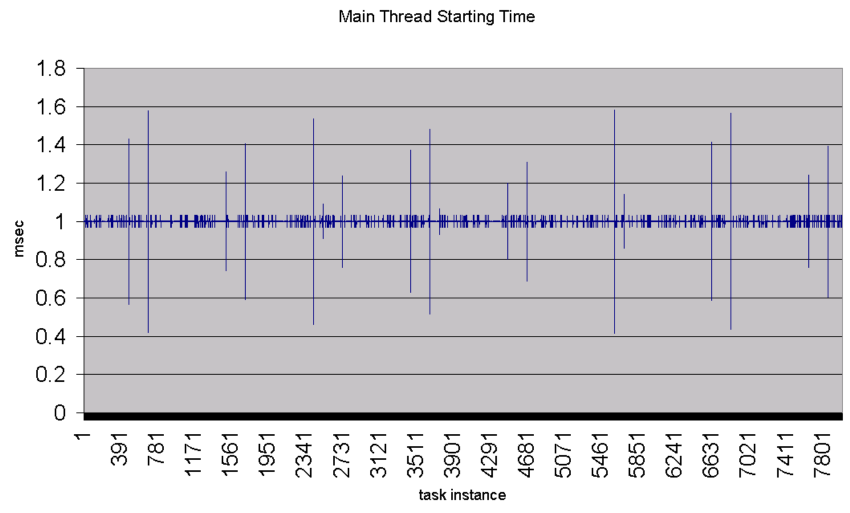


Fig. 4. Periodic task starting time, PC no name, loaded with copying files

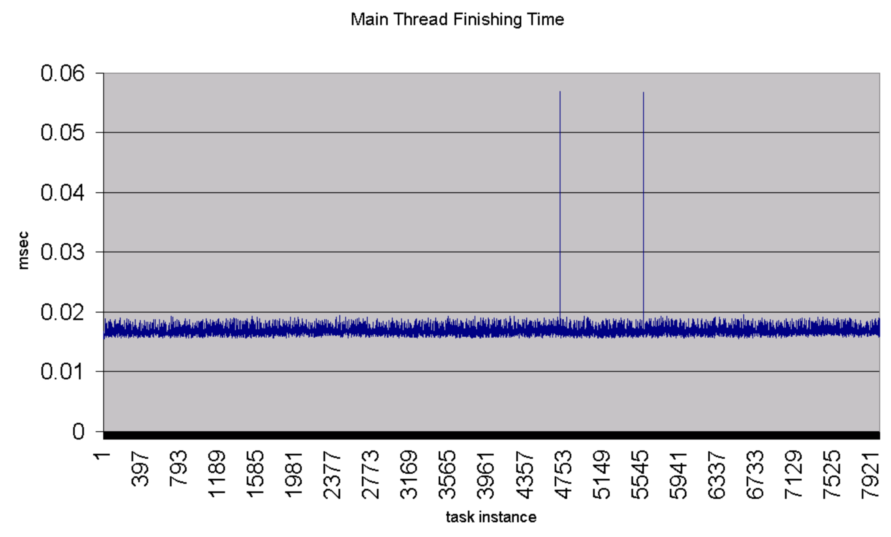


Fig. 5. Periodic task finishing time, PC Dell, loaded with copying files

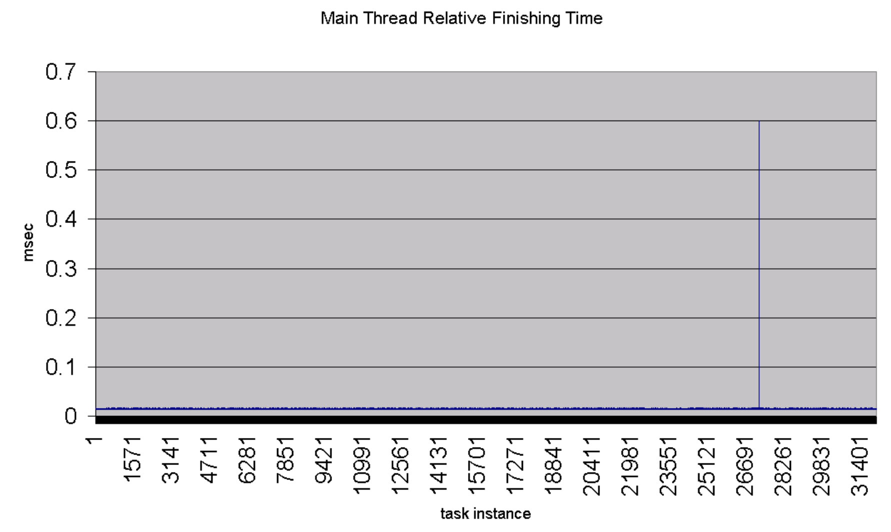
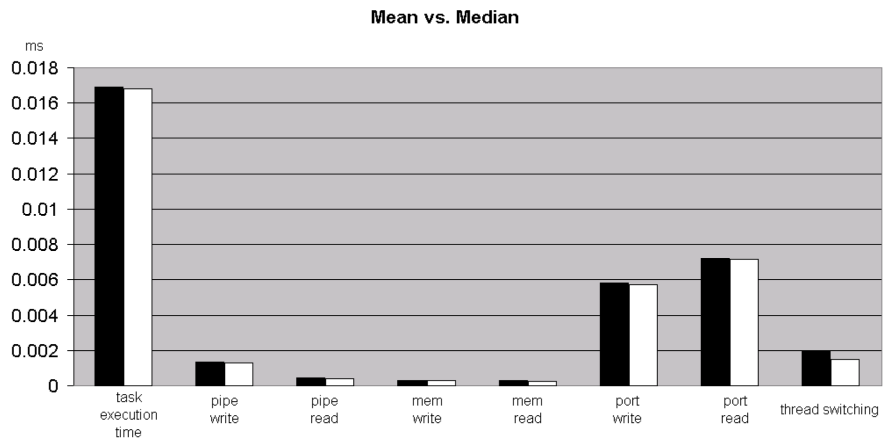


Fig. 6. Periodic task finishing time, PC no name, loaded with copying files

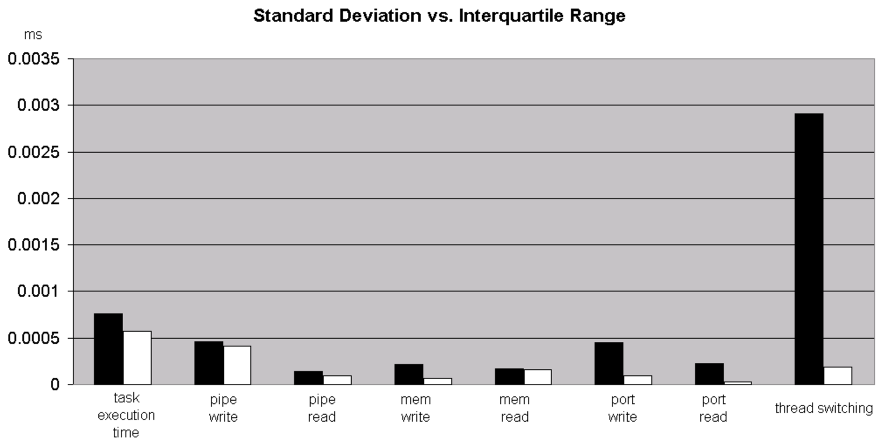
The graphs show that the RT-golem runs smoother on the Dell PC (Figures 3 and 5), than on the no name PC (Figures 4 and 6). To evaluate the outlying values, figures 7 and 8 show the mean vs. median comparison, as well as standard deviation vs. interquartile range comparison. It stems from definitions of mean and standard deviation, that they are more impacted by outlying values than median and interquartile range.



dark: mean

light: median

Fig. 7. Execution time means vs. medians. PC Dell, loaded with copying files



dark: standard deviation
light: interquartile range

Fig. 8. Execution time means vs. medians. PC Dell, loaded with copying files

6. Conclusion and future work

From the results one can conclude that the heavy hard disk operation [3] imposes more load on the system than the real-time tasks load. As the load resulting from hard disk operation is quite common in the system according to POSIX 1003-13 PSE 54 profile, it can be concluded, that

- the scheduler manages to handle more tasks than is was presumed by its authors,
- the Linux kernel operation significantly influences the real-time task jitters.

There is a question, whether the real-time characteristics of the system (the measured jitter spikes) could be smoother, if the Linux kernel were ported on a CPU designed for real time.

The hardware is built as a layered structure of basic hardware resources (disk, memory, processor registers, etc.), and following advanced means (instruction queues and priority rules). The advanced resources are basically the same as the resources used in operating system. These higher level hardware resources can be seen as a hardware implementation of the operating system resources.

A CPU designed for real time (DSP) has different architecture than a CPU designed for general purpose application. It is unlikely, that it could optimally support a full range POSIX 1003-1 compliant kernel.

Based on performed RTLinux and Linux kernel analysis, as well as on measured results, it can be reasonably concluded that for the RTLinux/Linux operating system, a general-purpose hardware is the optimal hardware platform.

Measurements performed at the level of a real-time task often provide valuable information on task jitters, but only little information on underlying causes. Therefore, it could be useful to create a small and simple HAL layer (module) in the Linux kernel, which intercepts timer interrupt and possibly other hardware means for a moment, and quickly gets and taps the diagnostic information needed. Another possible idea is, that such tool could be integrated into lower (architecture dependent) layer of the RTLinux HAL.

Acknowledgement

This work was supported by the Ministry of Education of the Czech Republic under Project 1M0567.

References

1. FSM Labs Inc.,*Getting Started with RT Linux,* 2001.
2. I. Ripoll, et al.: *WP1: RTOS State of the Art Analysis: Deliverable D1.1: RTOS Analysis,* OCERA, 2002.
3. Proc. SPIE Vol. 4563, Sensors and Controls for Intelligent Manufacturing II, Peter E. Orban, Ed., pp. 10-16, 2001.
4. C. Dougan, Z. Mwaikambo: *Lies, Misdirection and Real-time Measurements,* `http://www.rtl.com` 2004.
5. RTLinux v.3.1 source code, /usr/src/rtlinux-3.1/schedulers/rtl-sched.c