STANDART LINUX OS WITH REAL-TIME SUPPORT

Vladimir Germanov
Burgas Free University

Stanislav Simeonov
Burgas Free University

Abstract: *The goal of this paper is to present and compare two different Linux extensions for making Real-time Linux solutions. The idea is to use and adapt general purpose operating systems to provide a real-time environment which is comparable to specialized RTOS. In this paper Xenomai and RTAI are present.*

Keywords: *real-time, operating system, linux, xenomai, rtai*

INTRODUCTION

The fast evolution in the field of Computer Science and Microelectronics led to a broad variety of new device used in everyday life - MP3-players, personal digital assistants (PDA's), notebooks, and navigation systems are just a few examples of innovative product divisions.

The diversity of available devices and the complexity of these systems made it highly desirable to use the advantages of an operating system. It should be capable to offer applications as well as programmers an interface to the underlying architecture. A well-defined Application Programmers Interface (API) enables programming of applications based on this API without taking care of system-specific properties.

Many applications especially in the field of digital signal processing have to meet time constraints in order to work properly. Crunching sounds while playing music on an MP3-player, or delayed video playback should be avoided although these are just soft real-time applications. But there are also fields which require hard real-time constraints. Programs for controlling a space vehicle, or for observing the correct function of power plants, fall in this category.

Up to now, specialized commercial real-time operating systems (RTOS) have been used widely. Examples are VxWorks, which were used during the Mars Pathfinder mission, or pSOS, and LynxOS. The performance of these systems is of course very good, and they are perfectly suited to meet the requirements in hard real-time environments. However, shortcomings are for instance the price and the probably small user/programmer community.[1]

The idea now is to use and adapt general-purpose operating systems (GPOS) to provide a real-time environment which is comparable to specialized RTOS.

The focus of this paper is on the GPOS Linux. The question is how Linux can be extended to provide real-time capabilities. Windows is also a widely known GPOS. It can also be analyzed considering its usability for real-time applications, but this is beyond the scope of this work.

Limiting to Linux, this paper considers two Linux real-time distributions namely Xenomai, and RTAI.

REAL-TIME LINUX DISTRIBUTIONS

All of the presented systems follow the principles of using a specialized microkernel to achieve real-time performance. This microkernel is responsible for the core functions of an operating system like interrupt handling, scheduling, timer handling, and interprocess communication. On top of this microkernel several domains coexist, which underlie the microkernel scheduler. Each of the domains owns a fixed priority, which allows distinguishing between tasks of different qualities during scheduling. Usually, the hard real-time tasks are running with a high priority, while a very low priority is assigned to the standard Linux kernel. This means, every time a real-time process is runnable the Linux kernel can be preempted, thus meaning the standard Linux kernel plays the role of an idle-task.

Another function of the real-time kernel is interrupt handling. Usually, the standard Linux kernel deals with real hardware interrupts. But this is not possible anymore, as the real-time microkernel is working directly on the hardware now. Because of this, a technique called "Interrupt Virtualization" is introduced, which means that the real-time microkernel emulates hardware interrupts to the Linux kernel. In order to work together the standard Linux kernel usually has to be adapted to the underlying real-time-layer.

The presented real-time distributions must not be considered independently. Rather, they are based on and influenced by each other. The diagram in figure 1 gives an overview about the temporal connections between Xenomai, RTAI, and RTLinux.

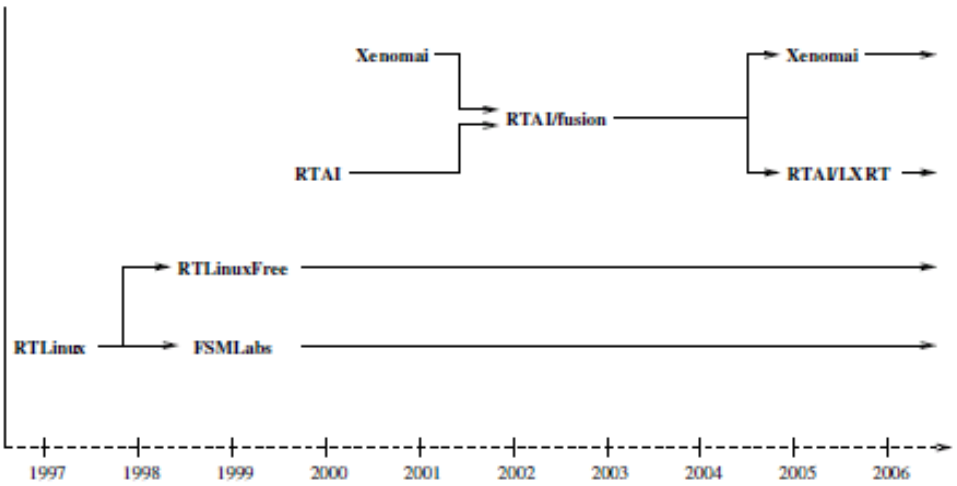

Fig. 1. Timeline of a real-time Linux distributions

The Xenomai project was launched in the year 2001. The head maintainers are currently Philippe Gerum, Bruno Rouchouse and Gilles Chanteperdrix. The main goal is to provide real-time capabilities based on some fundamental functionalities exported by an abstract RTOS core. Furthermore, mechanisms are provided to support applications which are originally intended to be used with traditional RTOS. In general, Xenomai aims on flexibility, extensibility, and maintainability rather than trying to achieve lowest technically feasible latencies like RTAI does. [6] The Xenomai kernel components underlie the GNU General Public License (GPL). However, it is still possible to distribute proprietary applications by using the User-Space libraries, which are under the GNU Lesser General Public License (LGPL).

When discussing the principles of Xenomai, it is necessary to have a closer look at the architecture. Figure 2 gives a good overview about the vertical structure of the overall system. Xenomai consists of several different abstract layers, which introduce an enormous flexibility.

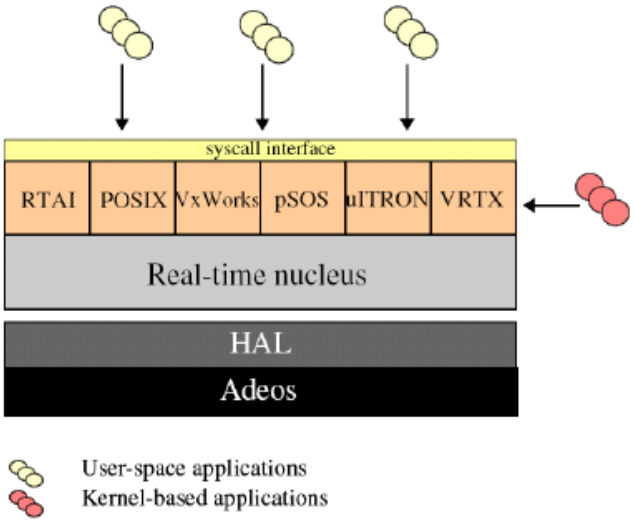

Figure 2: Architecture of Xenomai [7]

On top of the hardware the Adeos nanokernel is working. The hardware abstraction layer (HAL) lying above Adeos is the architecture-dependent part of Xenomai. When porting Xenomai to be runnable on another hardware platform, the HAL is the layer which has to be adapted, provided that Adeos is available for the new architecture. The central part of the system is the abstract RTOS nucleus which runs on top of the HAL. It implements a set of generic RTOS services, which are common for the most systems. The services can be either accessed through a Xenomai native API or via the different RTOS-API's build on top of the nucleus. These additional so called API skins for traditional RTOS make it possible to run even legacy software on top of the Xenomai core. This is especially useful when migrating to the Xenomai architecture because applications need not to be rewritten entirely. Xenomai enables users to run their applications either in Kernel- or in User-Space, though tasks living in the latter one have guaranteed but worse execution times. Running tasks in the User-Space context ensures the reliability of the system, because the crash of one process in Kernel-Space can lead to a breakdown of the entire system.

Despite the fact that Xenomai offers a wide variety of traditional RTOS API's there is also a native API, which can be used fully independently. It is implemented in the kernel module "xeno_native.ko" [7].

The followings are some important features of the Xenomai without commenting them in detail. Moreover, some key points are presented to make a comparison with other Linux real-time distributions possible. More precise information can be found in [7,8,9,10].

1. Multithreading support
2. Basic synchronization support
3. Timer and clock management
4. Basic memory allocation

The development of the *Real-time Application Interface (RTAI)* was started around the year 2000 by Professor Mantegazza at Dipartimento di Ingegneria Aerospaziale in Mailand. As already illustrated in figure 1, it is a fork from the RTLinux project.

RTAI is very similar to Xenomai as a consequence of its merging to RTAI/Fusion project in the year 2002. Many API calls of Xenomai's native API have their counterparts in the RTAI API, distinguishable only by their names. Nevertheless, RTAI does not follow the concept of offering multiple API's to the user, like Xenomai does. In fact, it is out of scope to provide the ability to run legacy RTOS applications over RTAI. As a consequence, there are not as many different abstraction layers stacked on top of each other as in the Xenomai case. Figure 3 gives an idea about the architecture of RTAI:

Like in Xenomai, RTAI regards the standard Linux kernel as the idle process with lowest priority, executed only when there are no other real-time processes schedulable. Timers in RTAI can be programmed to work either in periodic or in one-shot mode. The one-shot mode offers a finer granularity with slightly more overhead, caused by the reprogramming of the timer.

RTAI comes along with its own schedulers, in order not to be limited by the capabilities of the standard Linux scheduler. It offers the possibility to choose between two different schedulers, implemented in the modules rtai_sched.ko and rtai_lxrt.ko. These two approaches differ only in the type of objects they can schedule in Kernel-Space.[14] In User-Space only applications both schedulers are equal.

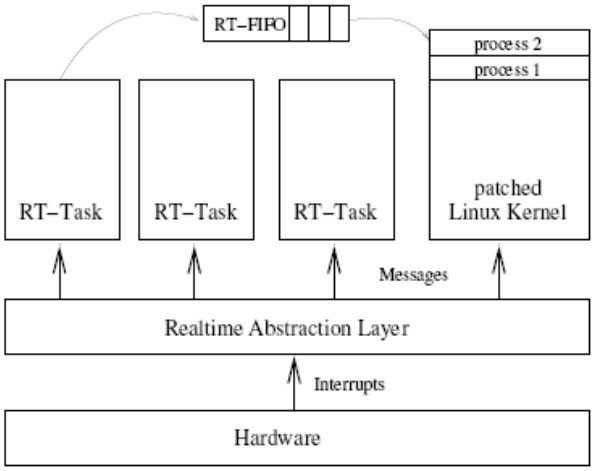

Figure 3: Architecture of RTAI

With rtai_sched.ko, specialized lightweight RTAI kernel threads can be scheduled as well as any schedulable Linux object like Linux processes, Linux threads and Linux kernel threads.

The purpose of the LinuX Real-time (LXRT) Scheduler in rtai_lxrt.ko is to offer a symmetric API to real-time RTAI tasks and Linux processes. User-Space applications based on this API can simply be transferred to hard real-time context by calling "rt_make_hard_realtime()". Basically, the usage of LXRT creates always kernel-threads which can then be scheduled by the real-time scheduler of RTAI. This allows achieving hard real-time performance with slightly worse execution times, caused by management overhead, than that of RTAI real-time tasks. Obviously, this is a great advantage when developing new real-

time applications. User-Space debugging tools are available and during the development process the code runs in a safe environment. After reaching a stable release the application finally may be migrated to a RTAI real-time task easily because of the symmetry of the LXRT API.

However, when making a Linux system call the User-Space application is migrated back to Linux and underlies the standard Linux scheduler. When the service request is fulfilled the task is handed back to the RTAI scheduler. Thus, using standard POSIX system calls should be avoided because the task will loose its hard real-time context. Instead, the RTAI API should be used in order not to be switched to Linux. Due to the fact, that the overall granularity of the Linux kernel steadily increases (see chapter 2), these switches will get less time consuming and more deterministic in future.

More detailed information about how the two different schedulers are working can be found in [14].

MESURMENTS

The purpose of this work is mostly to present a quantitative comparison of different Linux real-time extensions. Basically, this paper deals with external interrupt latencies and scheduling latencies. Other parameters which can give information about the quality of real-time capabilities are for example context switch times, process dispatch latencies or the time that is needed to allocate a certain amount of memory. Furthermore, the terms and conditions for tests have to be well defined e.g. architecture of the test system, system load, etc..

Test System - a standard desktop pc (x86) has been used for the measurements. The appropriate setup of the Linux distribution is much easier and suitable kernel patches are available. The test system consists of:

- Intel(R) Pentium(R) 4 CPU 2.80GHz
- 1024 MB main memory
- base kernel: 2.6.17 and 2.6.18
- 100 Mbit/s Ethernet

The real-time Linux distributions were used in the following versions:

- Xenomai 2.2.4, Adeos I-PIPE Patch "adeos-ipipe-2.6.17-i386-1.5-00.patch"
- RTAI 3.4, Patch "hal-linux-2.6.17-i386-1.3-08.patch"

The load on the x86 test system was applied in the following way:

- I/O-load: Flood-Ping to the test system and usage of "dd"
- CPU-load: CPU Burn-in v1.00 1

The measurement of the response time to external interrupts gives a good idea about real-time capabilities. The time between the occurrence of an external signal and the start of the corresponding interrupt service routine (ISR) is measured. Therefore, the authors used a parallel-port-loopback connector which was already used by Thomas Wiedemann in [13] for measuring interrupt latencies with plain Linux.

In a first cycle, latencies were measured over a standard Linux 2.6.17 kernel "CONFIG_PREEMPT" enabled in order to evaluate the improvements of preemptibility of the Linux kernel. Finally, the kernel module has been reeimplemented to work on top of the different real-time Linux API’s. All measurements were made according to an interrupt frequency of 1ms and under heavy stress, by means of I/O- and CPU-load. The results are illustrated throughout the figures 4 and 5.

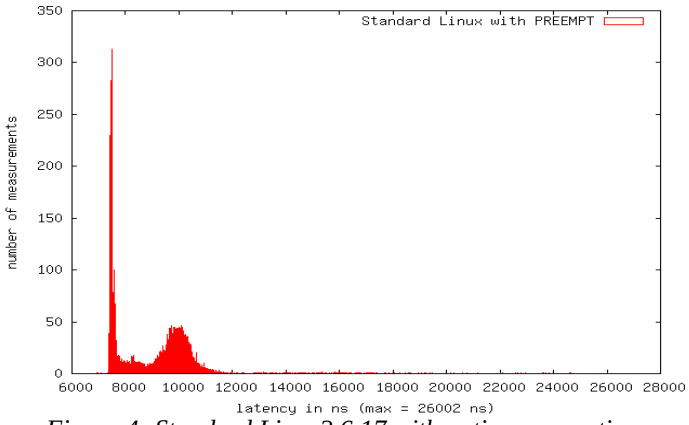

Figure 4: Standard Linux2.6.17 with native preemption

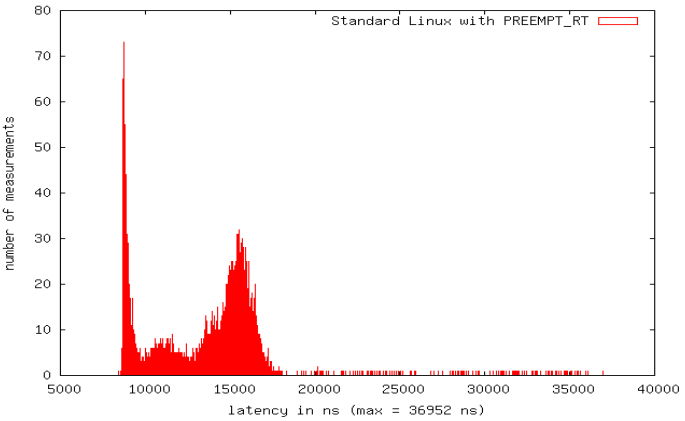

Figure 5: Standard Linux2.6.17 with real-time preemption patch

The figures 6 and 7 show the interrupt response times of Xenomai and RTAI. The maximum latencies are 95 % lower than standard Linux kernel with preemption patch.. This is the most important thing for real-time applications. Xenomai shows higher latencies if native preemptibility of the Linux kernel is activated, although the interrupt reply is initiated in interrupt handler context. The native preemption should only provide better results and only when a switch to the Secondary Domain occurs.

In a second step, a periodic task was set up and the jitter of activation times has been monitored. The standard Linux kernel was compiled with the standard timer accuracy of 250 Hz (4ms). However, the resolution is quite coarse grained for real-time applications. That's why the period of the task was adjusted to 10ms.

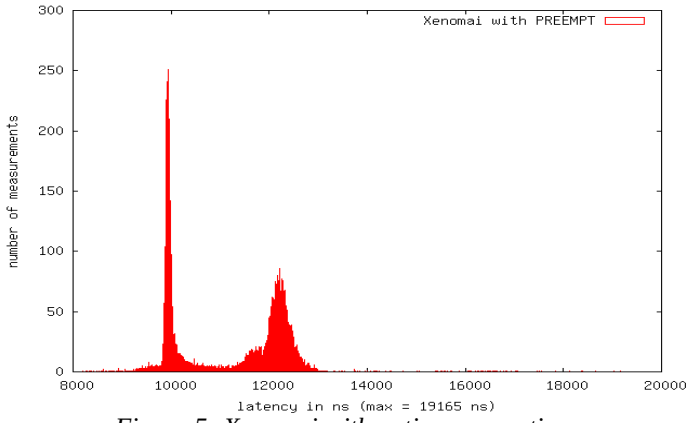

Figure 5: Xenomai with native preemption

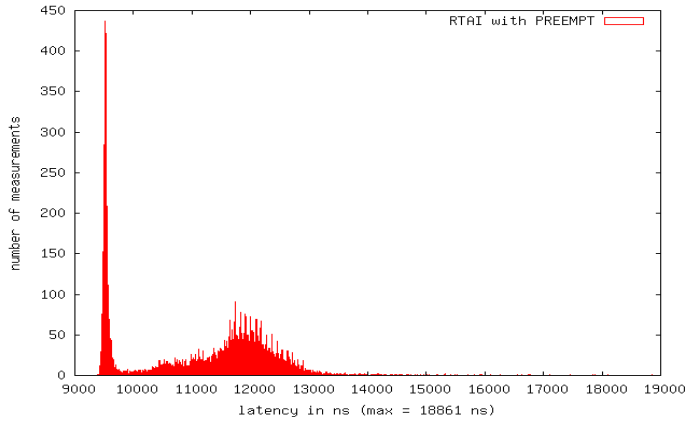

Figure 6: RTAI with native preemption

With Xenomai and RTAI the timer resolution is much more accurate, allowing setting up a task running with a period of 100µs. All measurements have been done over a period of 1 minute (600000 samples) and under heavy stress, by means of I/O- and CPU-load. The results are illustrated throughout the figures 7 to 10.

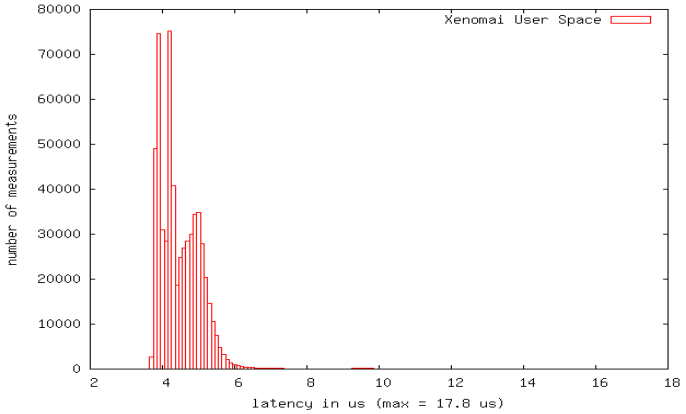

Figure 7: Xenomai User-Space Task

With Xenomai and RTAI much better results are achievable. The measurements with a periodic timer interrupt handler show the lowest scheduling jitter. The User- and Kernel-Space task is slightly worse but delivers also reasonable latencies. The results of the periodic User-Space task with RTAI are similar to its Xenomai counterpart. Both are nearly equivalent. Nevertheless, the measurements of the RTAI Kernel-Space task look quite impressive. Nearly all latencies are below 2µs which is much better than the Xenomai Kernel-Space task. A reason therefore could be the different implementation of the Kernel-Space latency task in RTAI..

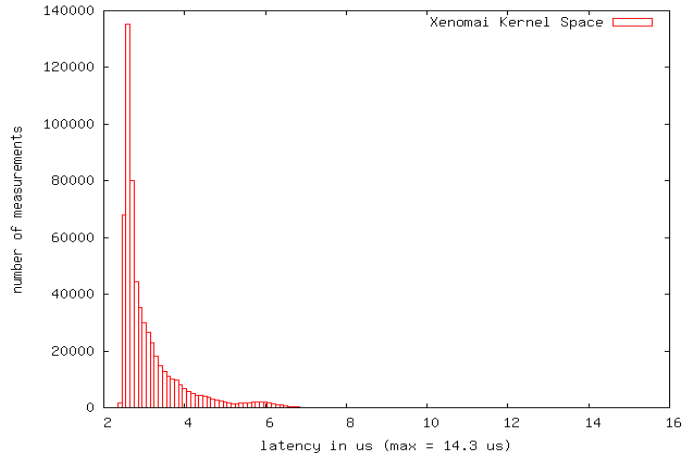

Figure 8: Xenomai Kernel-Space Task

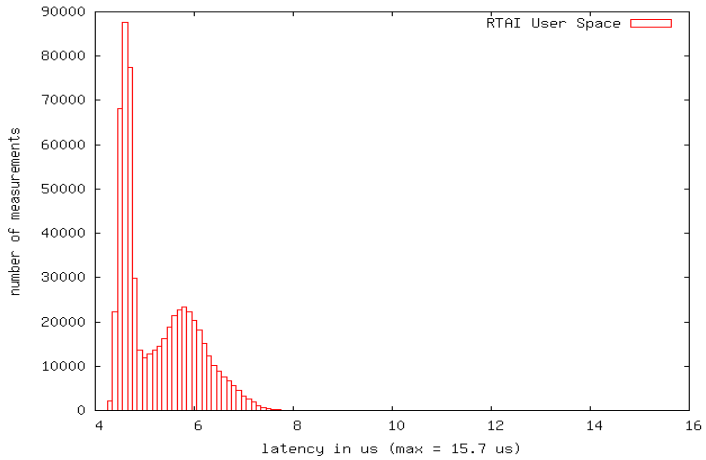

Figure 9: RTAI User-Space Task

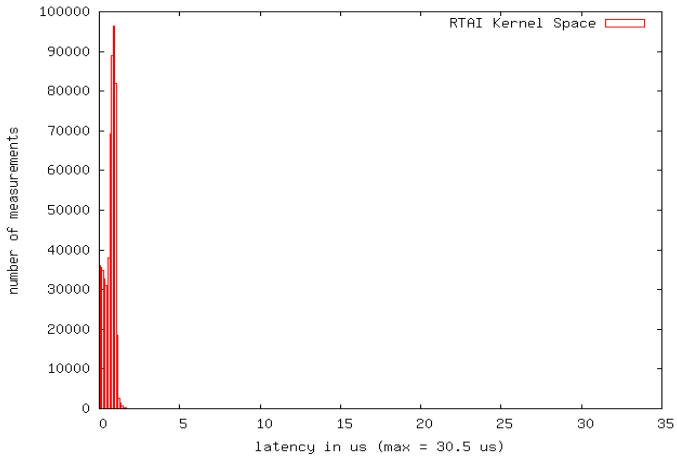

Figure 10: RTAI Kernel-Space Task

CONCLUSION

The paper has mainly discussed two major real-time Linux extensions. A short overview about the state-of-the-art development in RTAI and Xenomai has been given.

Xenomai and RTAI are maintained very well. The list of supported architectures is much longer and kernel patches are available for lots of kernel versions, either Linux 2.6 or 2.4. Xenomai and RTAI are also well suited to be used in closed source commercial products. The User-Space libraries are under the terms of the GNU Lesser General Public License (LGPL).

This paper illustrated some representative measurements of real-time capabilities.

The response time to external interrupts is very similar between Xenomai and RTAI. The results of the scheduling latency measurements are a bit di1fferent. In User-Space there is no difference between Xenomai and RTAI. However, if using a Kernel-Space task, RTAI provides much better performance than Xenomai. Having a look at the context switch times to Linux, Xenomai is slightly better than RTAI.

REFERENCE

[1] JaneW. S. Liu: Real-Time Systems Prentice Hall (2000). ISBN 0-13-099651-3

[2] Daniel P. Bovet, Marco Cesati: Understanding the Linux Kernel, 3rd Edition O'Reilly (2005). ISBN 0-596-00565-2

[3] Arnd C. Heursch, Alexander Horstkotte and Helmut Rzehak: Preemption concepts, Rhealstone Benchmark and scheduler analysis of Linux 2.4 published on the Real-Time & Embedded Computing Conference, Milan, November 27-28, 2001.

[4] ClarkWilliams, Red Hat, Inc.: Linux Scheduler Latency March 2002.

[5] http://lwn.net/Articles/146861/

[6] XenomaiFAQ: http://www.xenomai.org/index.php/FAQs

[7] Xenomai Native API Tour: http://www.xenomai.org/documentation/branches/v2.0.x/pdf/Native-API-Tour.pdf

[8] Xenomai Whitepaper: http://www.xenomai.org/documentation/branches/v2.0.x/pdf/xenomai.pdf

[9] Adam M. Costello and George Varghese: Redesigning the BSD Timer Facilities published in SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 28(8), 883–896 (10 JULY 1998).

[10] Marshall Kirk McKusick, Michael J. Karels: Design of a General Purpose Memory Allocator for the 4.3BSD UNIX Kernel published in Proceedings of the San Francisco USENIX Conference, pp. 295-303, June 1988.

[11] Karim Yaghmour: Adaptive Domain Environment for Operating Systems

[12] Life with Adeos: http://www.xenomai.org/documentation/branches/v2.0.x/pdf/Life-with-Adeos.pdf

[13] Thomas Wiedemann: Seminar Paper: How Fast Can Computers React date:21/12/2005

[14] RTAI 3.4 User Manual rev 0.3

[15] Karim Yaghmour: The Real-Time Application Interface date: published in 2001

[16] Michael Barabanov: A Linux-based Real-Time Operating System date: published in June 1, 1997

[17] Free Evaluation Version of RTLinux/Pro http://www.fsmlabs.com/order.html date: 20/02/2007