Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-25 (2016) 401–406

Real-time capabilities of Linux RTAI

J. Arm * **Z. Bradac** ** **V.Kaczmarczyk** ***

* *Department of Control and Instrumentation, Faculty of Electrical Engineering and Communication, Brno University of Technology, Brno, Czech Republic (e-mail: jakub.arm@phd.feec.vutbr.cz)*
** *Department of Control and Instrumentation, Faculty of Electrical Engineering and Communication, Brno University of Technology, Brno, Czech Republic (e-mail: bradac@feec.vutbr.cz)*
*** *Department of Control and Instrumentation, Faculty of Electrical Engineering and Communication, Brno University of Technology, Brno, Czech Republic (e-mail: kaczmarczyk@feec.vutbr.cz)*

Abstract: This article deals with the Linux real-time extension RTAI. This extension provides soft real-time capabilities in the user space and hard real-time capabilities in the kernel space. In this work, some key parameters of the RTOS are measured such as switching context time, periodic thread jitter, interrupt latency response, execution time of synchronization primitives and heap allocation time. This measurements create a test bench of real-time system capabilities which should help the developer to create truly real-time application on the specific hardware based on measured real-time capabilities. RTAI functions can be called from the user space using LXRT and also from the kernel space. These two spaces are compared from the point of view of the developer and real-time capabilities. All measurements have been done on the LinuxCNC 2.7 based on Debian 3.4 distribution running on AMD Athlon XP 3000+ uniprocessor. To accomplish these measurements, new benchmark application has been developed.

Keywords: Real-time, Linux, RTAI, LXRT, jitter, switching context time, response latency, function latency, memory allocation, benchmarking.

1. INTRODUCTION

Real-time operating systems are used in safety-critical, industrial, medical and automotive applications. Execution time of every operation has to be bounded in hard real-time systems and lowered in soft real-time systems. This OS is usually built in the embedded system.

The current challenge is to do more computational operations in lesser bounded time. Therefore more powerful devices like industrial PC are used as a control unit of the system. Because of the x86 architecture of the processor, some x86 RTOS has to be used which is usually at the commercial grade. There are some open-source projects which tries to have the same or better performance, i.e. Linux RTAI, Xenomai, chinese RT-thread, RTEMS, MarteOS, etc. On the other hand, there are commercial RTOS like QNX, VxWorks or LynxOS also based on POSIX standard which are often compared to.

Reliability of RTAI module is achieved by using in the large community. Some mentioned RTOS are also alive long time that proclaims also their reliability. The main parts of the RTOS are deterministic preemptible scheduler, task manager, memory manager, timer manager and HAL (Hardware Abstraction Layer). These parts are implemented as another Linux kernel which runs the whole default kernel and passes data from hardware (see Fig. 1).

In this article, a performance of Linux RTAI is measured. The switching context time is quantified using two threads synchronized via mutex. Mutex release time, semaphore release time, real-time FIFO operation time and change priority time are also measured to get the worst execution time. One of the most important parts of RTOS is the precise timer. So the jitter of the periodic timer is measured with and without load. Operations with the memory are crucial in the RTOS. The memory allocation is considered to be not deterministic. However, Linux RTAI has some modification to support real-time allocation. It implements even the new TLFS (Two-Level Segregated Fit memory allocator) published in Masmano et al. (2004). Memory allocation time and memory operation time are measured.

All parameters are measured in the kernel space by a created kernel module and also in the user space using LXRT (LinuX Real-Time). Using LXRT, threads are executed in soft real-time mode by default. But these threads can be executed also in hard real-time mode by passing them directly to RTAI module.

2. RELATED WORK

In [Barbalace et al. (2007)], four Linux systems are compared. Interrupt response time, rescheduling and response time using UDP packet via RTnet are measured. All measurements are done in the kernel mode so potential MMU remapping has no impact. A result of the article is that Linux RTAI performance is as good as VxWorks and Xenomai performance or even better specially the communication via RTnet.

10.1016/j.ifacol.2016.12.080

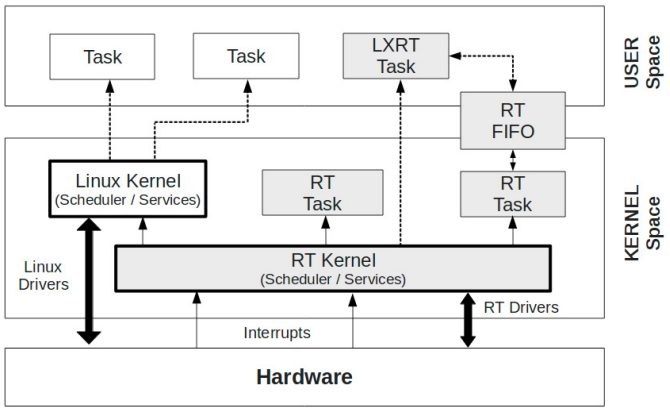

Fig. 1. Linux RTAI [Salazar (2011)]

In [Cereia and Scanzio (2012)], the jitter of the periodic timer are measured on Linux RTAI using developed IgH EtherCAT kernel module. Minimal and maximal values of the jitter are measured with and without skew compensation of the timer drift. Values are measured in the kernel space and also in the user space with and without the I/O load. Some function latencies are also measured. Authors proposed some new method to lower the user space maximal value of the jitter with the I/O load. Originally, the user space application communicated with the kernel space EtherCAT master module using the Linux character device. This method caused greater time drift in the user space task especially with an I/O load. This impact is decreased using a shared memory and two semaphores for the synchronization.

In [Koh and Choi (2012)], the periodic task jitter and some execution times of synchronization function are measured. These measurements are executed on the Intel Core 2 Duo at $3,00\ GHz$ running Linux kernel 2.6.32. A comparison between user and kernel space is also done. The main comparison is between Xenomai and RTAI which concludes in slightly lower execution times in Xenomai kernel space.

In [Brown and Martin (2010)], jitter of a periodic task and interrupt response time is measured in user and kernel space. These values are evaluated for mainline Linux, Linux with PREEMPT-RT patch and Xenomai. The comparison is done for values in user and kernel space. The measure unit is a part of device and is based on AVR MCU. This benchmark is done on a device from Beagleboard family based on ARM architecture. Some high measured values were discarded. As a result, Xenomai patch has the best values when the kernel space is used. The interesting part of this work is that synchronization primitive execution times of the mainline Linux are the smallest. This might be caused by the absence of real-time capabilities and by using raw non-preemptible spinlocks in the kernel.

In [LinuxDevices Staff (1997)], a new scheduler is approached that executes soft real-time task even if there are hard real-time tasks using the Constant Bandwidth Server algorithm that acts as another hard real-time task and executes soft real-time tasks. This makes the soft real-time tasks more responsive.

3. LXRT

LXRT is an extension of RTAI service to the user space. It provides API functions similar to RTAI kernel API where are some small modifications, e.g. little different function calls or automatic global registering of objects. The overall system architecture using LXRT is on Fig. 2. Advantage is the better safety of kernel space against possible malware application. On the other hand, this is disadvantage because an application does not have access to raw drivers. The biggest advantage is that a debugger can be used to bug searching. Both implementations need to load RTAI kernel modules like `rtai_hal` and `rtai_sched`.

According to POSIX, Linux threads are created using `pthread_create()`. Using `rt_task_init()`, the thread is considered as a soft real-time thread scheduled by the standard Linux scheduler. A process can be set to the scheduler using SCHED_FIFO policy that preempts a thread in case of IO operation, higher priority ready task or calling `sched_yield()`. Using SCHED_RR policy, threads can be preempted after the certain time quantum by a thread with the same priority. Executing `rt_make_hard_real_time()` makes a thread scheduled by RTAI FPPS (Fixed Priority Preemptive Scheduling) scheduler.

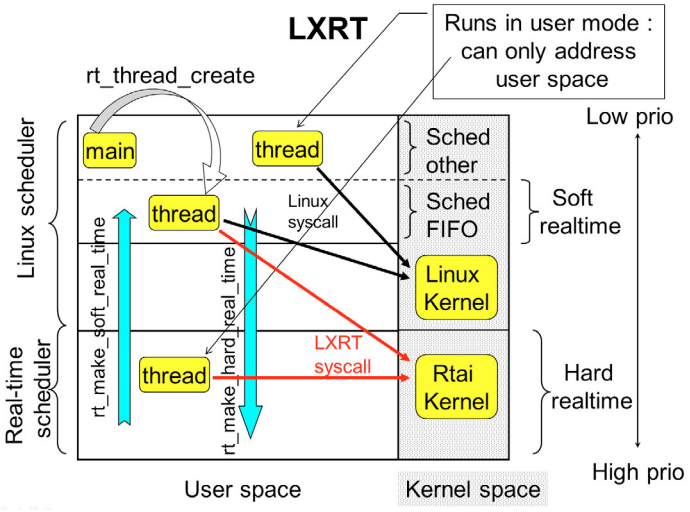

Fig. 2. LXRT

4. MEASUREMENT

In this section, all measurements are presented. Measured parameters are jitter of the periodic task, interrupt response time latency, switching context time, mutex execution time, semaphore execution time, real-time FIFO execution time, memory allocation time and memory operation time. The measurements are performed directly on the machine, results are evaluated and graphical outputs are made. To simulate embedded environment without video output, measured values can be sent by ethernet and evaluated on a server like in [Slanina and Srovnal (2007)] and [Slanina and Srovnal (2008)].

This benchmarking takes long time and a lot of values are measured. So the internal highest precision timer, that is APIC (Advanced Programmable Interrupt Controller), is

used to automate this task. This timer ticks at frequency of $20,749,242\ Hz$. Every call, the `rt_get_time_ns()` gets the new current value. The result of each measurement consists of minimal value, maximal value, average value and corrected sample standard deviation calculated using the equation 1. Deviation values can be extended by a coefficient $k_p = 3$ to achieve a probability of the right value $99,7\ \%$ to provide the correct range.

$$\sigma = \sqrt{\frac{1}{N-1}(\sum_{i=1}^{N} x_i^2 - N\overline{x}^2)u_A} = \sqrt{\frac{1}{n(n-1)}\sum_{i=1}^{n}(x_i - \overline{x})^2} \quad (1)$$

Measurements are performed on system idle and under simulated load. This load is created by the user space soft real-time thread of a period of 1 μs which does a lot of work in memory. The thread is scheduled under SCHED_FIFO policy with a priority lesser than threads for measuring. When the priority is higher or even the same, the measuring thread is not executed because of low resources.

On this occasion, the measuring application RTcap has been developed to execute and process all measurements. So the measuring is automated.

Hardware and software in this measurement : PC - AMD Athlon XP 3000+ with the CPU frequency $2,2\ GHz$ uniprocessor, $1,5\ GB$ RAM, 333 MHz bus frequency, 32 *bit* data width, 64 kB L1 cache and 512 kB L2 cache; OS - LinuxCNC 2.7 Wheezy supported by RTAI 5.0 based on Debian with kernel 3.4.55 SMP PREEMPT i686.

4.1 Periodic task jitter

A measurement of the periodic task jitter is one of the most used benchmark to get real-time capabilities of a system. In LinuxCNC distribution, there is built-in application that measures the periodic thread jitter of period of 1 ms and 25 μs in the nanosecond resolution called HAL Latency Test. Or the famous Cyclictest can be used to get a jitter value.

The HAL Latency Test gives the results listed in Table 1.

	Max interval [ns]	Max jitter [ns]
Servo thread (1 ms)	1049079	50479
Base thread (25 μs)	76471	51506

Table 1. HAL Latency Test results

RTAI module provides LXRT service which enables registering real-time task from the user space. There are two types of user space real-time threads : hard (set by `rt_make_hard_realtime()` function) and soft real-time thread (default settings). The last choice is to make real-time thread in the kernel module. The timer provided by RTAI HAL has two modes of operation : *one-shot* and *periodic* mode. In periodic mode, a timer period is written once to the auto-reload register and there are not any other operations done by HAL. So this mode is more computational effective.

All measurements have been done with a timer period of 1 ms. Tables 2, 3, 4 and 5 summarize measured values of the one-shot and periodic timer without and with any other computational load. The jitter of a lesser period is also measured using a period value of 25 μs and the results are in Tab. 6, 7, 8 and 9 with load. The count of each measurement is $1,000,000$. The jitter value is represented by the maximal period deviation.

In the periodic mode, the maximal deviation is the smallest. On the other hand, the oneshot mode has smaller deviation. The results are comparable to [Cereia and Scanzio (2012)] when periodic mode of timer is used. Comparing to [Koh and Choi (2012)], our measurement shows lesser maximal jitter values in periodic and oneshot mode even under load. Moreover, our measurement has been done many times in compare with the cited one and with smaller period. The cited measurement was performed on Intel Core2 Duo at 3.00 GHz.

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1000.000	1.082	86.717
User hard RT	1000.000	0.162	19.465
Kernel	1000.000	0.155	16.083

Table 2. Measurement of the one-shot timer period without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	999.987	1.974	240.526
User hard RT	999.987	1.145	5.337
Kernel	999.988	0.821	3.410

Table 3. Measurement of the periodic timer period without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1000.000	1678.034	89426.320
User hard RT	1000.000	0.229	12.619
Kernel	1000.000	0.263	14.235

Table 4. Measurement of the oneshot timer period with load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	999.987	1.251	76.665
User hard RT	999.987	0.776	4.373
Kernel	999.988	0.477	2.446

Table 5. Measurement of the periodic timer period with load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	25.000	1.004	269.817
User hard RT	25.000	0.116	16.250
Kernel	25.000	0.146	14.800

Table 6. Measurement of the lesser one-shot timer period without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	25.013	1.319	134.042
User hard RT	25.013	1.237	2.953
Kernel	25.013	0.700	3.917

Table 7. Measurement of the lesser periodic timer period without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	25.000	302.792	93081.430
User hard RT	25.000	0.157	16.105
Kernel	25.000	0.262	13.381

Table 8. Measurement of the lesser oneshot timer period with load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	25.013	4.180	2877.272
User hard RT	25.013	0.508	3.794
Kernel	25.013	0.211	1.989

Table 9. Measurement of the lesser periodic timer period with load

4.2 Interrupt response latency

A measurement of the interrupt response latency is also one of the most used benchmark to get real-time capabilities of a system. In the Linux kernel, there is built-in measurement that results as a maximal value of the interrupt response latency. On the other hand, RTAI HAL implements its own IDT (Interrupt Description Table) by Adeos core to get the response and provides IDT for the kernel using pointers (Ipipe technology) to process its IRQ (Interrupt Request).

In Linux, interrupt are served according to the interrupt stack. Hardware interrupts are served at first. After that, softirqs are served. Then, scheduled tasklets are served. The tasklet is a job planned by fast hardware interrupt or softirq to do more work and be prepared to another possible interrupt. After all, the interrupted process is resumed.

The interrupt is registered using `rt_request_global_irq()` on S6 (pin 10) of LPT1 and can be cleared using `rt_free_global_irq()`. The measurement is automated and many times repeated so the trigger signal is generated using a periodic thread which sets D7 (pin 9) of LPT1 that is directly connected to S6. Some non-negligible time consumes `outb()` functions which is measured as a value of $(1.199 \pm 0.012)\mu s$. Next time part is consumed by `rt_get_time_ns()` function which gets the current time using APIC (Advanced Programmable Interrupt Controller) timer. This time is measured as a value of $(0.114 \pm 0.008)\mu s$. These measurement was performed $1,000$ times. Only the first measured time is subtracted from results.

Using the mainline preemptible kernel, interrupt response latency values should by comparable. A value of interrupt latency is measured only in the RTAI kernel space because there is pour implementation of interrupt servicing by LXRT because of hardware character of demanded service. The measurement is in Tab. 10.

The maximal response latency is smaller compared to a measurement in [Brown and Martin (2010)] where Xenomai module is measured.

	Average [μs]	Deviation [μs]	Max [μs]
Kernel	8.197	0.709	25.515
Kernel (load)	7.717	0.725	21.525

Table 10. Measurement of the interrupt response latency

4.3 Switching context time

According to definition, a context switch is a operation that ends the execution of a thread and starts the execution of another thread. This operation consists of context saving, picking up a thread, context loading and start of a new thread execution. This process also includes erasing of the processor pipe-line and refreshing of the TLB (Translation Lookaside Buffer) which translates the logical addresses fast using a table.

The switching of a thread context is one of the most frequent operation of any OS. In RTOS, its bounded execution time is important. To get more performance of an OS, the context switch has to be as short as possible. RTOS needs to be deterministic so the picking thread algorithm has to behave according to defined rules which has to choose the only one thread.

Linux RTAI is time and event based. So the scheduler performs an operation on scheduler tick and when some system operation like `sem_wait()`, `rt_set_task_priority()`, or `sem_signal()` occurs. The principle of this measurement lies in the synchronization of two threads using a semaphore, when the higher priority threads waits for the semaphore signal. The negligible latencies of executions of synchronization calls are omitted.

In Tab. 11, there is a measurement of $1,000,000$ times context switch on system idle. In Tab. 12, there is a similar measurement with the load.

The average of this measurement is low because of fast searched schedule solution which lies in finding one thread to execute among three threads. In the maximal value, a systematic error of other prioritized interrupt execution is evinced. This error can be reduced using statistical measurement methods. High values do not mean that the switching context takes long but there are other interrupts and tasks run meanwhile. So the measured value means precisely how long it takes to switch from one thread to another using synchronization primitive at the specific priority and environment from the point of view of the application.

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	3.495	0.492	126.095
User hard RT	1.141	0.075	16.433
Kernel	0.344	0.022	11.728

Table 11. Measurement of the switching context time without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	3.153	70.734	50,020.113
User hard RT	0.983	0.060	16.766
Kernel	0.290	0.029	13.254

Table 12. Measurement of the switching context time under load

4.4 System function execution time

The execution time of system functions is not usually measured. It is considered to be as small as possible. These values are significant when these functions are used many times. The most important is that these values are

bounded under any circumstances. The function execution times are measured by measuring of a time of 100,000 function executions. The assumption of thread execution without any preemption is needed to be fulfilled. For this reason, the measurement can not take long time because of overall system unresponsiveness. In the soft real-time user space, a preemption can occur during thread execution so these results can not be accurate. The measurement is performed while system is mostly idle (see Tab. 13 and 15) and while system is under load (see Tab. 14 and 16). The measurements are performed for functions `rt_sem_signal()` and `rt_mbx_send()` which are the most used synchronization primitive function calls.

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1.214	0.522	103.337
User hard RT	0.567	0.042	11.910
Kernel	0.185	0.018	1.675

Table 13. Measurement of the `rt_sem_signal()` without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1.079	0.168	12.315
User hard RT	0.590	0.025	2.955
Kernel	0.119	0.016	1.529

Table 14. Measurement of the `rt_sem_signal()` under load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1.611	0.438	50.852
User hard RT	0.756	0.065	5.709
Kernel	0.533	0.044	4.620

Table 15. Measurement of the `rt_mbx_send()` without load

	Average [μs]	Deviation [μs]	Max [μs]
User soft RT	1.356	0.211	12.591
User hard RT	0.808	0.058	7.622
Kernel	0.344	0.048	4.688

Table 16. Measurement of the `rt_mbx_send()` under load

4.5 Memory allocation time

A memory allocation on heap is considered to be non-deterministic. Therefore, it is recommended to allocate all used memory at startup. LinuxCNC locks almost all RAM memory at startup to be allocated. However, there is an approach how to make memory allocator deterministic so called TLSF (Two-Level Segregated Fit) allocator. According to the experiments in Masmano et al. (2004), the execution time is bounded in stated test cases and even faster than other methods. The stated measurements was done on a similar machine to our machine.

In Figs. 3, 4 and 5, there are a 1,000 times measurements of allocation per each memory size over allocating chunk of memory. All measurement were performed under load to achieve the worst case values. A memory is allocated and freed using `malloc()` and `free()` functions. There are even RTAI special functions `rtai_malloc()` and `rtai_free()` but its execution time is rising with the allocated size because it uses `vmalloc()` which allocates memory in the virtual memory. Using `malloc()`, the average execution time is nearly constant up to the maximal size of RAM memory. The opposite situation is in the kernel mode where functions `rtai_kmalloc()` and `rtai_kfree()` are used instead of functions `kmalloc()` and `kfree()` to achieve nearly constant function of execution time over allocated size up to the maximal size of RAM memory.

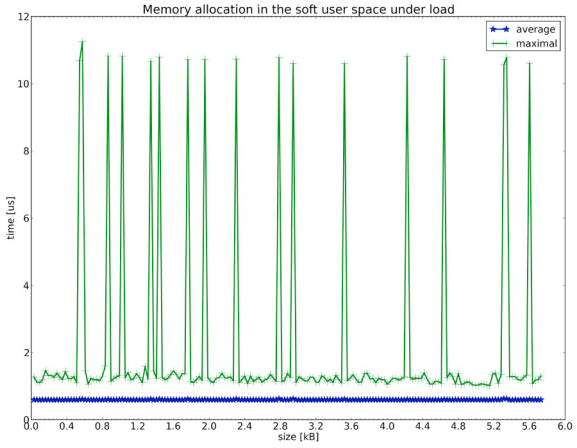

Fig. 3. Measurement of memory allocation execution in the soft RT user space

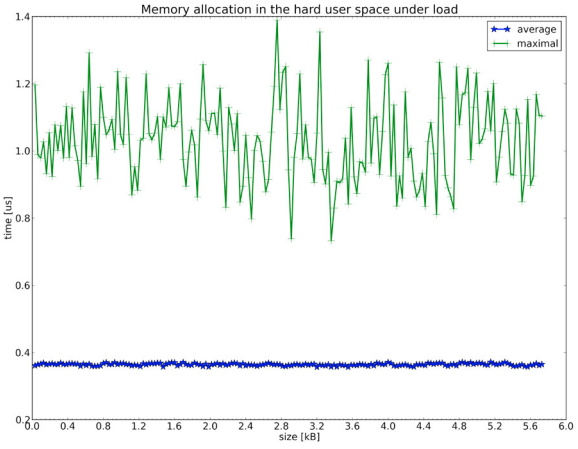

Fig. 4. Measurement of memory allocation execution in the hard RT user space

5. CONCLUSION

Some of the main parameters of RTOS like switching context time, periodic task jitter, interrupt latency time, synchronization function execution time and memory allocation time have been measured. These parameters have been measured using soft real-time LXRT, hard real-time LXRT and in the kernel space. Some measured values are comparable or even better to other measurements and measurements using Xenomai patch. According to the results, RTAI patch can be used for a task requiring submilisecond precision. The worst case precision

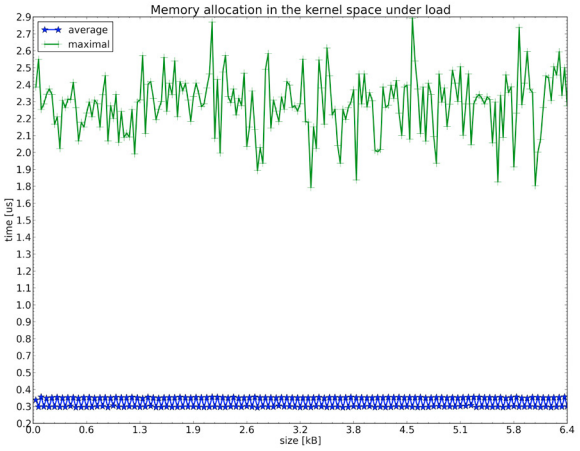

Fig. 5. Measurement of memory allocation execution in the kernel space

of the periodic jitter was measured in the kernel space as a value of 2.446 μs under load. This work also shows that Linux RTAI can meet its deadlines even on older uniprocessor architecture.

This benchmarking gives the developer a better view of real-time capabilities of the RTAI and LXRT on the given machine to meet system requirements. According to this, an application can be created to meet real-time capabilities. The kernel space capabilities are comparable to the hard real-time user space using LXRT. An advantage of LXRT is that a debugger can be used.

Execution times of synchronization functions are comparable. The boundedness along with deadlock free architecture is important. The maximal values are not determinant because this measurement does not take long enough. Another factor is that RTAI is open-source so it is without any warranty. On the other hand, source code is included and some basic behaviour is tested.

ACKNOWLEDGEMENTS

This work was supported by grant No. FEKT-S-14-2429 - "The research of new control methods, measurement procedures and intelligent instruments in automation", which was funded by the Internal Grant Agency of Brno University of Technology.

REFERENCES

Barbalace, A., Luchetta, A., Manduchi, G., and Moro, M. (2007). Performance comparison of vxworks, linux, rtai and xenomai in a hard real-time application. *Real-Time Conference, 2007 15th IEEE-NPSS*, 1 – 5. doi: 10.1109/RTC.2007.4382787.

Brown, J. and Martin, B. (2010). How fast is fast enough? choosing between xenomai and linux for real-time applications. *Twelfth Real-Time Linux Workshop on October 25 to 27, Nairobi*.

Cereia, M. and Scanzio, S. (2012). A user space ethercat master architecture for hard real-time control systems. *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. doi:10.1109/ETFA.2012.6489584.

Koh, J.H. and Choi, B.W. (2012). *On Benchmarking the Predictability of Real-Time Mechanisms in User and Kernel Spaces for Real-Time Embedded Linux*, chapter Computer Applications for Security, Control and System Engineering, 205–212. Springer Berlin Heidelberg. doi:10.1007/978-3-642-35264-5_29.

LinuxDevices Staff (1997). Improving the responsiveness of linux applications in rtai. *LinuxDevices Archive*. URL `http://linuxdevices.linuxgizmos.com/4th-rtl-workshop-improving-the-responsiveness-of-linux-applications-in-rtai/`.

Masmano, M., Ripoll, I., Crespo, A., and Real, J. (2004). Tlsf: a new dynamic memory allocator for real-time systems. *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, 79 – 88. doi: 10.1109/EMRTS.2004.1311009.

Salazar, M. (2011). Cove project and p3-dx units. [online]. URL `https://marcelorsalazar.wordpress.com/author/marce00/`.

Slanina, Z. and Srovnal, V. (2007). Embedded linux scheduler monitoring. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, 760–763. doi:10.1109/EFTA.2007.4416851.

Slanina, Z. and Srovnal, V. (2008). Embedded systems scheduling monitoring. In *Systems, 2008. ICONS 08. Third International Conference on*, 124–127. doi:10.1109/ICONS.2008.47.