AUTOMATYKA • 2006 • Tom 10 • Zeszyt 2

Marcin Piątek*

Real Time Application Interface focused on servo motor control

1. Introduction

The GNU/Linux operating system consists of the Linux kernel and GNU software. The latest is responsible for development process and system usability. It was first released in September 1991 by its author Linus Torvalds. His goal was to develop an operating system compatible with the POSIX standard. GNU/Linux is released under General Public License what means that everyone can download the source code and develop his own changes as long as one distributes them with the same restrictions. These two facts: the compliance with POSIX and GPL license, demonstrate GNU/Linux advantages: maintainability and usability. In the year 2006 a great number of developers has been engaged to the GNU/ Linux project. New features, device drivers and a compliance with standards have been implemented [1].

GNU/Linux introduces two features interesting and important to our examining. Communication between the Linux kernel and the user space processes is achieved by polling system calls – the set of functions implemented in the kernel. As parameters the names of the files grouped into the `/dev` folder are passed. These files are abstraction of hardware e.g. floppy, console or serial port. So if we would like to send some data to a serial port we can simply open the `/dev/ttyS0` file and write these data as it would be a regular file. A device driver implemented into the kernel takes care of all other operations. In conclusion, the process executed at the user space communicates with the kernel space via the `/dev` filesystem. It is shown at Figure 1.

The second important feature of the GNU/Linux system is a possibility of broadening kernel features by loading a kernel module. To add some new feature to the Linux kernel we can write a new code, compile it, and load it into the kernel with the `insomd` command. The new feature is available just right after loading. It means that we can test the new kernel software without reloading the OS kernel. We can also easily unload the module from the kernel by using the `rmmod` command [4].

Despite all this features the GNU/Linux operating system cannot meet hard real time requirements due to its architecture. Yet it is open source, an experienced developer can

* AGH University of Science and Technology, Cracow

introduce arbitrary changes to the system. First attempt was considered in 1997. Since then many different solutions with such a goal have been successfully implemented and deployed. They are available as a patch to a regular kernel source – the file includes only differences between the regular source and an improved one. It is easy to manage it with an appropriate software.

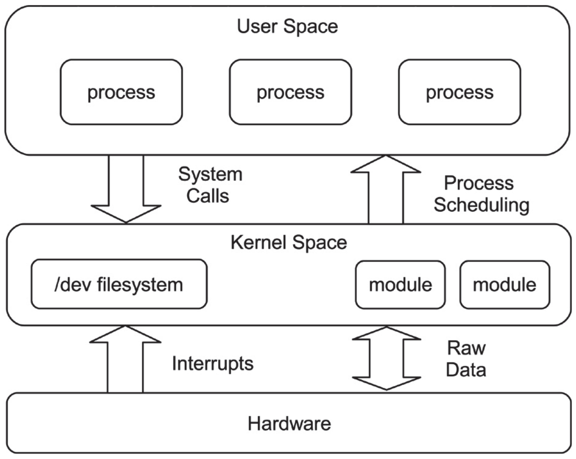


Fig. 1. Communication between the kernel and user spaces

Considering the way of changing the GNU/Linux operating system we can classify these patch sets into four groups:

1) Micro Kernel Architecture,
2) POSIX Real-Time Extension,
3) Nano Kernel Architecture,
4) Resource Kernel Extension.

The exact description can be found in [5].

2. Real Time Application Interface

Real Time Application Interface Project has been started at Technical University in Milan. Simultaneously to this paper the last stable version 3.3 is available. It supports several processor architectures:

– x86,
– x86_64,
– PowerPC,
– ARM (StrongARM, ARM7).

As long as RTAI is considered Micro Kernel Architecture is used [8]. Basically it is characterized by a very small kernel – micorkernel which lies between Linux and computer hardware layers. It takes care of hardware interrupts and hardware-software communication to reach the hard real time restrictions. It also reimplements the regular kernel schedule function to add real time tasks and the real time processes service. It is important to notice that RTAI implements both: the hard real time tasks in the kernel space and the hard real time processes in the user space. Communication between real time tasks and the user space is possible through FIFO-like devices (the files in the `/dev` folder). The real time task can be performed as a kernel module, what is of a primary importance during development process. The RTAI system architecture is shown on Figure 2.

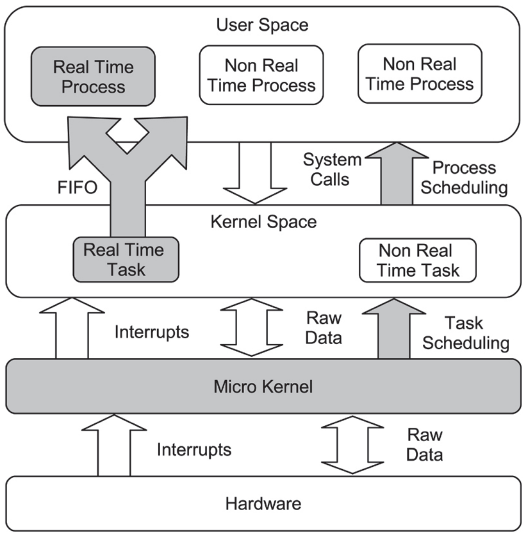


Fig. 2. Real Time Application Interface structure schema

Elements introduced or modified by a RTAI developers group are marked with grey color. A similar but not identical interface can be used to manage real time tasks and processes. C or C++ (only in the user space) language can be used during the development process. The interface is quite advanced – satisfactory even to demanding engineer. RTAI can be extended with the RTAI-Lab project which includes:

– collaboration set to Scilab/Scicos,
– collaboration set to Matlab/Simulink®,
– Comedi project integration set,
– graphical monitoring application,
– net_rpc communication set (host and target).

Advanced multilayer control algorithms can be easily developed due to the RTAI structure. Direct control should be placed in the hard real time kernel task, optimization and adaptation algorithms in the hard real time process and visualization and user interface should be implemented as the regular GNU/Linux process.

3. Latency and jitter

The system is deliverd with the testing software. It is easy to check the basic real time systems paramters: latency and jitter. The latency is defined as a time delay between an event request and the start of the event service routine. The jitter is a possible scatter of the periodic event begining time [5]. Both definitions are illustrated at Figure 3.

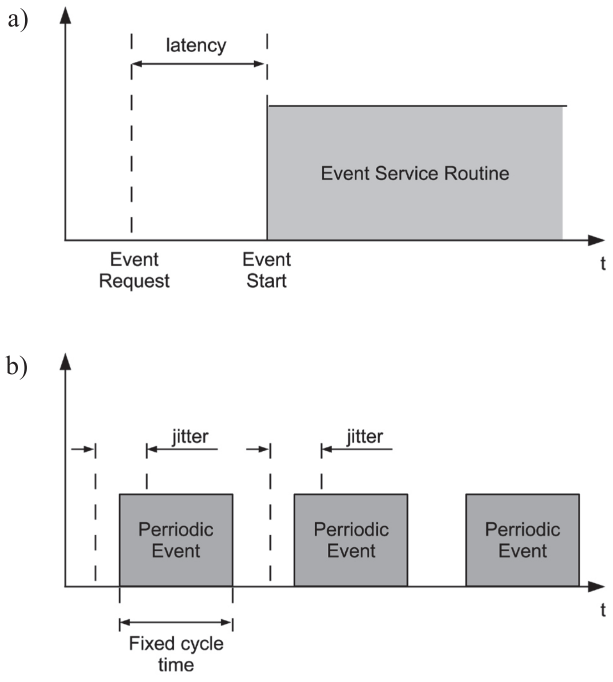


Fig. 3. Latency (a) and jitter (b)

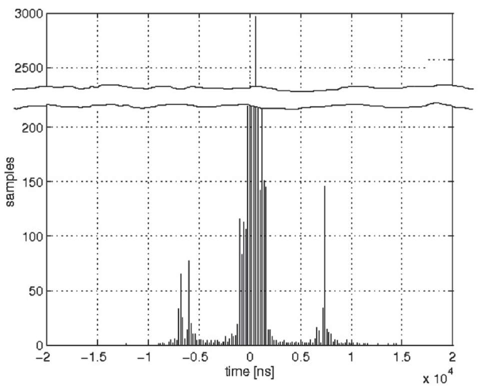
These two parameters are collected with the almost empty and heavy loaded system with the hard real time kernel task and the hard real time user space process. Additionally the jitter collection is done in a case when two tasks or two processes are running in the system. One of them is faster and it has higher priority then the other, what is the reason of the frequent preemption. The results are shown in Table 1.

Unfortunately these tests cannot be used with other operating systems. To collect appropriate results the testing system is built. It consists of the Personal Computer and the modified RT-DAC4/PCI multi I/O board equipped with hardware counters. Time can be measured very accurately due to the board. During tests the jitter values are collected for the GNU/Linux-RTAI kernel task and for Matlab/Simulink® with the Real Time Windows Target extension [10]. During the tests the hard real-time timer is set to call task once for 1 ms. Histograms are depicted for 10000 samples each. The results are shown at Figures 4 and 5.

Table 1
RTAI parameters: latency and jitter

	Avg. latency	Max. jitter	Fast task max. jitter	Slow task max. jitter
Kernel task not loaded system	–910 ns	9533 ns	16218 ns	15056 ns
Kernel task heavy loaded system	–427 ns	16421 ns	17924 ns	19455 ns
User process not loaded system	–404 ns	10696 ns	26697 ns	31866 ns
User process heavy loaded system	140 ns	20286 ns	24001 ns	33503 ns

a)



b)

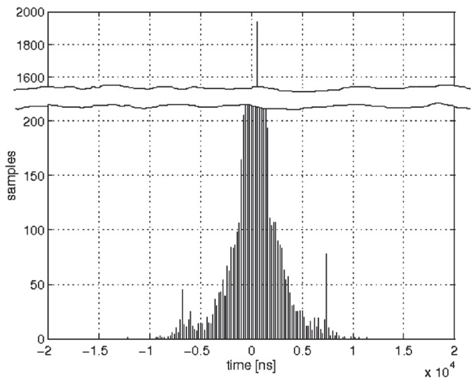
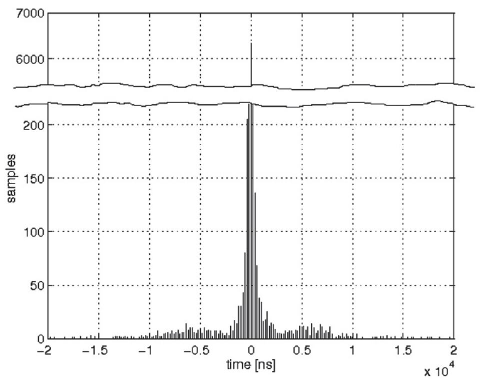


Fig. 4. Jitter histogram while the GNU/Linux-RTAI operating system is: a) not loaded; b) heavy loaded

a)



b)

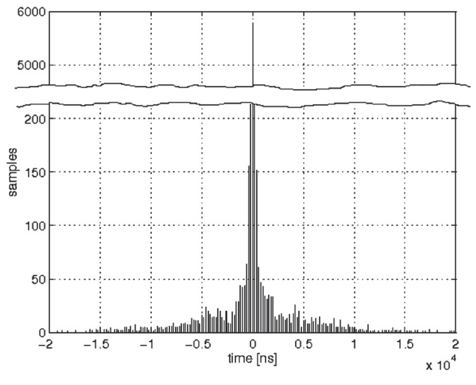


Fig. 5. Jitter histogram while the Matlab/Simulink® with the Real Time Windows Target extension is: a) not loaded; b) heavy loaded

The most important adventage of the GNU/Linux-RTAI operating system is the total lack of the long, random occuring delays. The values of the maximum jitter during tests are shown in Table 2.

Table 2
The maximum jitter

	Value [ns]
RTAI – not loaded system	12200
RTAI – heavy loaded system	14400
RTWT – not loaded system	280800
RTWT – heavy loaded system	533400

4. Generating functional signals

To generate some functional signals a testing system is built. It consists of a Pentium Celeron 1 GHz microprocessor based Personal Computer, the RT-DAC4/PCI multi I/O board, the GNU/Linux operating system and the RTAI software.

The jitter parameter (tested in previous section) appears while a square wave is generated. To generate this signal a simple hard real time task is written and executed. The task consists of a periodic event declaration and its definition. In each event request output signal changes to opposite digital value (0, +5 V).

Figure 6 presents a square wave with the period 50 µs. The jitter does not exceed 15 µs.

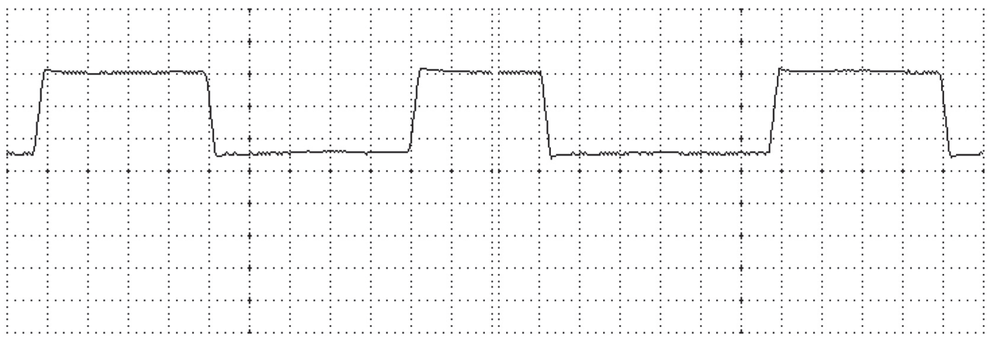


Fig. 6. The generated square wave – oscilloscope screen. Voltage 2 V/div *vs*. time 10 µs/div

Another well known problem is to use floating point arithmetic in the kernel space task. It is obvious that in the kernel space a Floating Point Unit is not used and even not initialized [4]. This problem has been solved by RTAI developers. Moreover FPU features can be easily applied. An interface permits to add and to use the standard C math library. Figure 7 illustrates these phenomena. It shows a sine wave generated by the hard real time task.

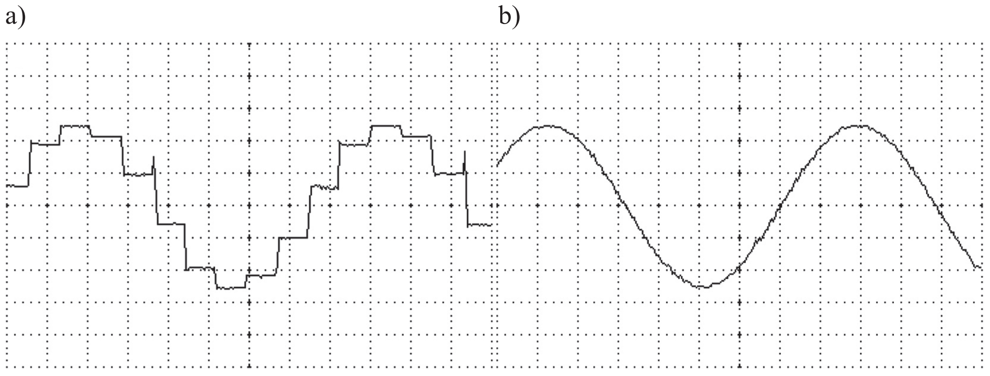


Fig. 7. The generated sin wave – oscilloscope screen:
a) Voltage 2 V/div *vs.* time 50 μs/div; b) Voltage 2 V/div *vs.* time 0.5 ms/div

5. Servo motor control system

As a next step, the servo motor control system is developed at the Personal Computer equipped with RT-DAC4/PCI multi I/O board. The adaptive algorithm is used and it is implemented with:

– GNU/Linux and Real Time Application Interface,
– Mirosoft Windows® XP and Matlab/Simulink® and Real Time Windows Target.

The control algorithm is divided into two parts: the direct proportional controller as the first layer and the Linear-Quadratic algorithm as the second layer [9]. In the case of the RTAI system, the direct control layer is implemented as the real time kernel task and the second layer as the real time process. The output visualization and the preset value collection are implemented as the regular GNU/Linux process. This is illustrated at Figure 8. The control period for the first layer equals to 1 ms and for the second layer equals to 10 ms.

The entire algorithm is implemented as a one layer with Matlab/Simulink® and the Real Time Windows Target.

For a Linear-Quadratic algorithm, cost function is defined as follows

$$J(u) = \int_0^\infty \left(x^T W x + u^T R u \right) dt \tag{1}$$

where

$$W = \begin{bmatrix} W_1 & 0 \\ 0 & W_2 \end{bmatrix} \tag{2}$$

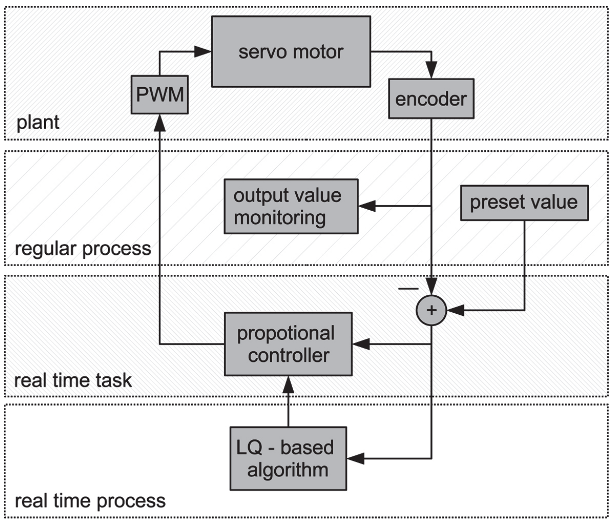


Fig. 8. Servo motor control system

In the developed algorithm W_1, W_2 and R are defined as:

$$W_1 = W_2 = 4\ln\left(|e|\right) + 0.1, \quad R = 1 \tag{3}$$

where e is a current error signal value. It means that parameters of the proportional controller are changing according to error change. The shape of function (3) causes decreasing of the proportional parameters when the error is close to 0. Due to that fact the whole system is less sensitive when the error is close to 0, particularly the control signal is less disturbed. It can be measured by a variance of the control signal. The pseudorandom number generator with variance equals to 10^{-6} was used to simulate disturbances. The numerical value of the variance are shown in Table 3. Figure 9 shows the response of the regular LQ algorithm for the step function. Figure 10 shows the response of the adaptive algorithm for the same input.

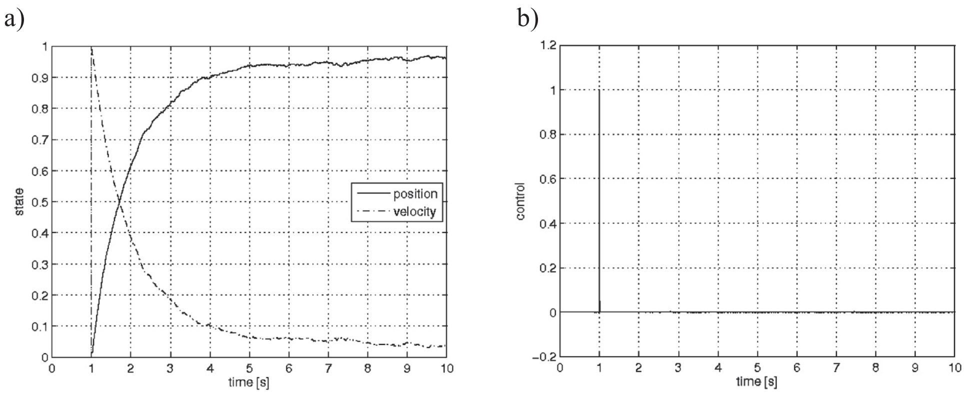


Fig. 9. Regular LQ algorithm results, state (a) and control figures (b)

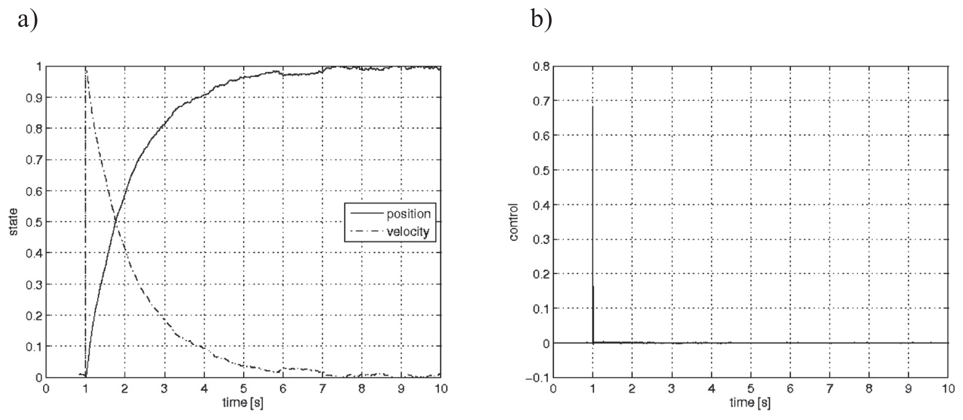


Fig. 10. Adaptive algorithm results, state (a) and control figures (b)

Table 3
The control variance values

	Control variance
RTAI – LQ algorithm	0.709214
RTAI – adaptive algorithm	0.361628
RTWT – LQ algorithm	0.71706
RTWT – adaptive algorithm	0.403795

6. Summary

During experiments the Real Time Application Interaface parameters are collected: the latency and the jitter. The jitter value is compared with Matlab/Simulink® and Real Time Windows Target results. Some basic tests are done. Square wave and sine wave are generated with the hard real time tasks due to C math library usage. The servo motor control system is developed with the GNU/Linux and the Matlab/Simulink® with the Real Time Windows Target. The results are illustrated in the graphical form.

References

[1] Love R.: *Linux Kernel Development.* Helion, 2004 (translated to Polish)
[2] Stevens W.R.: *Advanced Programming in the UNIX Environment.* WNT, 2002 (translated to Polish)
[3] Raymond E.S.: *The Art of Unix Programming.* Helion, 2004 (translated to Polish)
[4] Corbet J., Rubini A., Kroah-Hartman G.: *Linux Device Drivers, third edition.* O'Reilly, 2005
[5] National Institute od Standards and Technology: *Introduction to Linux for Real-Time Control.* NIST, http://www.isd.mel.nist.gov/projects/rtlinux/, 2002

[6] Proctor F.M., Shackleford W.P.: *Real-time Operating System Timing Jitter and its Impact on Motor Control.* SPIE Conference, 2001

[7] Dozio L., Mantegazza P.: *Linux Real Time Application Interface (RTAI) in Low Cost High Performance Motion Control.* A conference of ANIPLA, Associazione Nazionale Italiana per l'Automazione (National Italian Association for Automation), Milano, Italy, 27–28 March 2003

[8] Racciu G., Mantegazza P.: *RTAI 3.3 User Manual.* https://www.rtai.org/index.php, 2006

[9] Mitkowski W.: *Stablilization of Dynamical Systems.* Kraków, AGH, 1996 (in Polish)

[10] Mrozek B., Mrozek Z.: *Matlab and Simulink, User Manual.* Helion, 2004 (in Polish)