УДК 004.415+004.94 DOI: 10.30977/BUL.2219-5548.2018.83.0.107

# TIME RESTRICTION ASPECTS IN THE MODELING OF CYBER-PHYSICAL SYSTEMS FOR INDUSTRY 4.0

**Lizárraga M. L.[1], Buelna A.[1], Díaz-Ramirez A.[1], Amaro-Ortega V.[1], Kostikova M. V.[2], González-Navarro F. F.[3], Werner F.[4], Burtseva L.[3]**

**[1]Tecnológico Nacional de México/IT de Mexicali, Mexico,**
**[2]Kharkiv National Automobile and Highway University, Kharkiv, Ukraine,**
**[3]Universidad Autónoma de Baja California, Mexicali, Mexico**
**[4]Otto-von-Guericke University Magdeburg, Germany**

***Abstract.*** *Cyber-physical systems in Industry 4.0 are comprised of a computational core, a communication core, and a physical system. The computational core has strict timing restrictions and represents a Real-time system. In this paper, the modeling of the computational core is discussed.*

***Key words:*** *Real-time system, Cyber-physical system, Scheduling, Industry 4.0, Earliest Deadline First policy, Rate Monotonic policy.*

## Introduction

Throughout history there have been different revolutions in industry, each of these revolutions representing a major change in the manufacturing process. The fourth industrial revolution, known as Industry 4.0 (I40), refers to a set of technologies aimed to create the concept of smart factories. Among these technologies appeared Cyber-physical systems (CPS), Internet of Things (IoT), Cloud Computing and Cognitive Computing. CPSs are considered to be a core technology for I40.

CPSs are physical and engineered systems, whose operations are monitored, coordinated and controlled by a computing and communication core [1]. They are smart networked systems with embedded sensors, processors and actuators that are designed to sense and interact with the physical world (including the human users), and support real-time, guaranteed performance in safety-critical applications [2].

The CPSs cover a wide range of possible areas in which they can be used. Applications include medical systems, assistance systems for elders, traffic control systems, robotic systems, control systems and automation of industrial processes, automotive safety systems and drive assistance, military systems, to mention a few examples.

One of the main features of a CPS is that they have strict timing restrictions, which must be satisfied, since otherwise, the results may be catastrophic. A system, which requires a complete assignment of the resources and provides functioning in a timely manner, is referred to as a Real-time system (RTS) [3]. An RTS interacts with the asynchronous calls to maintain a continuous relationship and remains synchronized with the environment, reacting opportunely to changes in the settings [4]. The design of such systems permits an opportune response and the execution of a task within predefined time constraints. From a functional viewpoint, an RTS is a computer system, which is dedicated to monitoring of a process or to the control of tasks.

The RTSs implemented in traditional hardware are used in a wide range of applications. They are embedded into devices of common use, which are known nowadays as the IoT, as well as into communication devices, such as mobile phones and computers. These systems have been a subject of great interest due to the trend towards the automation of quotidian use systems and applications. Examples are self-driving cars, autonomous airplanes, sensors and robots to care the elderly, to mention some of them [5].

Modeling software for I40 involves modeling software for the computational core of a CPS, which is an RTS. Due to the nature of its temporal restrictions, a particular operating system, known as a real-time operating system (RTOS), and a specific Application Programming Interface (API), must be used. PREEMPT-RT, a Linux variant, and a Portable Operating System Interface (POSIX), are among the most popular RTOS and API for modeling and implementing real-time systems, respectively.

In this paper, the modeling of software for I40 is discussed. An introduction to RTSs is presented, along with a review of tools to model and implement them.

The rest of the paper is organized as follows. In Section 2, we introduce theoretical aspects of RTS and RTOS along with the Rate Monotonic (RM) and Earliest Deadline First (EDF) scheduling policies. Section 3 discuses the key aspects to model the computational core of a CPS. Section 4 shows an example. Finally, in Section 5 some conclusions and future work are discussed.

## RTSs

An RTS is a computational system that not only must provide accurate results, but also, it must provide them in a timely manner. Modeling this kind of systems represents a challenge since the temporal constraint must be identified and controlled. Additionally, a designer must verify that all the temporal restrictions will be met before implementing the RTS. As mentioned previously, the computational core of a CPS is in fact an RTS.
An RTS is conformed of a set of applications that request access to processors and resources. The scheduling algorithms control the access to processor and resources. Task scheduling is probably the most intensively studied area in the RTSs, since the most important feature of this type of computer systems is to ensure that all the tasks comply with their temporary restrictions. In a monoprocessor system, only one task can be executed at a time; any other task has to wait until the processor is free and can be re-scheduled. Nowadays, the goal of multiprogramming is to be able to have several tasks continuously running, in order to maximize the use of the processor. Figure 1 generalizes the structure of an RTS.

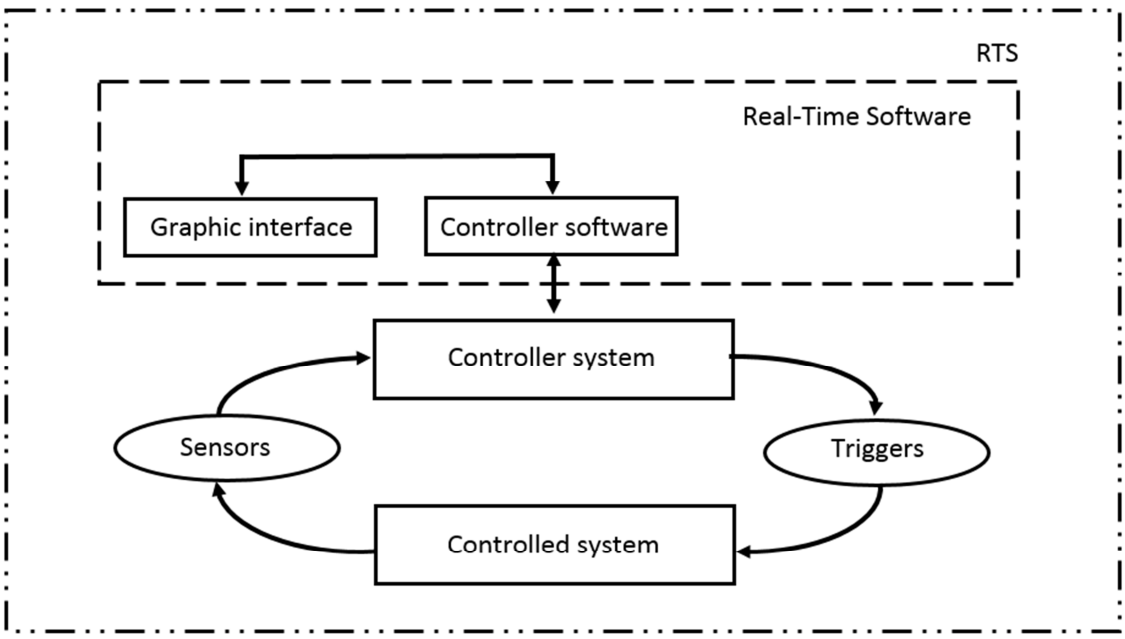


Fig. 1. Components of an RTS

## Definitions

An RTS is commonly comprised of tasks, where each task is subjected to a series of temporary restrictions. The number of processors limits the maximum number of tasks that can be executed simultaneously on a computer. Therefore, it is necessary to define which tasks have to be executed at each instant of time on each processor. The algorithms for this assignment are defined by the dispatching rules, also called planning policies. In the planning context, a task in an RTS represents a set of related jobs that provide some functions of a system. Every job in a task is released periodically, sporadically, or aperiodically. The majority of the restrictions imposed to an RTS are expressed by the task release times, execution times, and most importantly, by the deadlines.

A task has three possible states. When it is running, it is called an *active task*. A task waiting for the processing is called a *ready task*, and all tasks waiting for the processor are kept in a waiting list called *queue of ready task*. The

planner chooses the order of executing the tasks based on the policy established by the selected scheduling algorithm.

Usually, the restrictions that are imposed on an RTS refer to the task deadlines. In an RTS, the maximum possible time necessary to obtain a response must be less than or equal to the task deadline. The value that the task contributes to the system depends on the fulfillment of the deadline, starting at the activation moment. The deadlines can be classified as follows:

- *Soft deadlines*: the consequences of a task that does not finish its execution before its deadline do not jeopardize the integrity of the system. Commonly, the value that the task contributes to the system is maximal if it is executed before its term, and it is decremented proportionally to the time of completion after the term.

- *Hard deadlines*: if the task does not finish its execution before the deadline, the integrity of the system is not committed, and the value that the task contributes to the system is zero in this case. These deadlines are similar to non-strict deadlines, with the difference that, if the task does not meet its deadline, the system is not able to continue the execution.

The tasks of an RTS are commonly classified into two basic models: periodic and sporadic tasks. In both models, the tasks are referred to as an infinite sequence of activations called jobs [6]. In the periodic task model, the arrival of the jobs of a task is strictly periodic, separated by a fixed interval of time, called period. In the sporadic task model, each job in a task arrives at any time once a minimum interval of time has elapsed since the previous job of the same task has occurred. Among the RTS models, the most popular is the periodic task model.

The RTSs may be classified into two categories, accordingly to the kind of their task's deadlines:

- *Soft RTS*: missing a task deadline produces a performance degradation, but the tasks may continue the execution, while the system tries to minimize the consequences of missed deadlines;

- *Hard RTS*: missing a task deadline is not acceptable due to possible catastrophic consequences.

A task is subjected to a series of time restrictions, as it is shown in Figure 2.

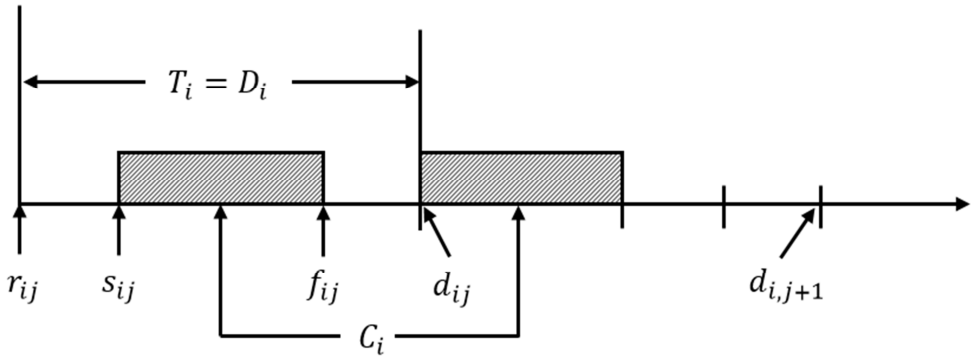


Fig. 2. Basic parameters of a task $\tau_i$: $r_{ij}$, $d_{ij}$, $s_{ij}$ and $f_{ij}$ are the release time, absolute deadline, start time and finish time of the task $i$ in the activation $j$, respectively; $C_i$ is the execution time in the worst-case (WCET); $D_i$ is the relative deadline; $T_i$ is the size of the activation period

**RTOSs**

An RTS must be executed under an RTOS, which is integrated into several modules that together allow the applications to interact with the hardware, in the same manner as a general-purpose operating system. However, an RTS must be designed in such a manner that the accomplishment of the timing restrictions of the tasks that comprise the system are assured by the selection and the use of an RTOS and real-time scheduling algorithms. The correct choice of an RTOS is a fundamental aspect in the design of an RTS.

Almost all existing RTOSs provide two priority-based scheduling policies: First-In First-Out (FIFO) and Round Robin (RR). Liu and Layland [7] introduced the RM and EDF policies for scheduling periodic tasks in hard RTSs. The RM algorithm assigns priorities inversely proportional to the task periods; it is an optimal static priority assignment policy. It can be mapped through the FIFO policy. The EDF policy assigns the highest priority to the job with the earliest deadline; it is an optimal dynamic priority assignment policy [8]. The EDF algorithm is not available in most of the

existing hard RTOSs. However, EDF gets a bet-better utilization of the available processor capacity, which allows the execution of more tasks in the same processor.

One of the most popular general purpose operating systems is Linux, which is employed by the academic community and companies for the development and execution of applications in diverse fields, such as control, computation, health, military and space, to name a few (see, e.g., [9]). Linux is not an RTOS. Its scheduling policies offer some level of timing guarantees for the soft RTSs, but they are not sufficient for systems with hard real-time constraints [10]. To use Linux as a base for an RTOS, some modification to its kernel must be made.

Generally, there exist two approaches to allow Linux to provide a hard real-time support: 1) at the hypervisor level [11, 12] and 2) at the OS scheduler level [13, 14]. The first approach allows the coexistence of both, an RTOS and a generic OS, where the first one has a higher priority comparing with a non-RTOS. In contrast, in the OS scheduler level, the real-time capabilities are provided using multiple scheduling classes, where each class has several scheduling policies, being a real-time scheduling class of the highest priority. This approach is employed by several projects, both as commercial and as open source ones. One of them is the RT-Preempt [14], which includes free open source patches.

**Architecture of an RTOS**

In an RTS, activities are commonly implemented as tasks or threads. An RTOS provides three important functions to attend the tasks: 1) scheduling, 2) dispatching, and 3) intercommunication and synchronization.

The scheduler defines the sequence of the jobs to be executed on the processor, selecting from the list of ready tasks, the next task to be executed. The scheduler implements the scheduling policies. In an RTOS, the scheduler is a fundamental component since the selected scheduling algorithm is in charge of accomplishing the temporal restrictions of the RTS. The dispatcher is a module that performs the necessary bookkeeping actions to start the execution of the chosen task. Intercommunication and synchronization services assure that the tasks cooperate correctly, process the actions to avoid a racing or similar anomalies.

To guarantee the time restrictions of the RTS, the scheduler implements real-time scheduling algorithms, such as RM, EDF, and synchronization protocols, such as PIP. The RTS designer chooses, among the available scheduling algorithms, those that better satisfy the RTS characteristics. Some examples of the commercial and free-software RTOSs are given in
Table 1.

Table 1 RTOS projects

| Project | Hard RTS | POSIX | GLIBC | Thread policy | Source |
|---|---|---|---|---|---|
| SCHED_DEADLINE | + | - | - | - | [10, 15, 16] |
| Litmus-RT | - | - | + | + | [13] |
| ChronOS | - | + | + | + | [17] |
| Xenomai | + | + | + | - | [18] |
| RTAI | + | + | - | - | [19] |
| RT-Linux | + | + | - | - | [11] |
| VXworks | + | + | - | - | [20] |
| OSE | + | + | - | - | [21] |
| MaRTE | + | + | - | - | [22] |
| SCHED_EDF | + | + | + | + | [23] |

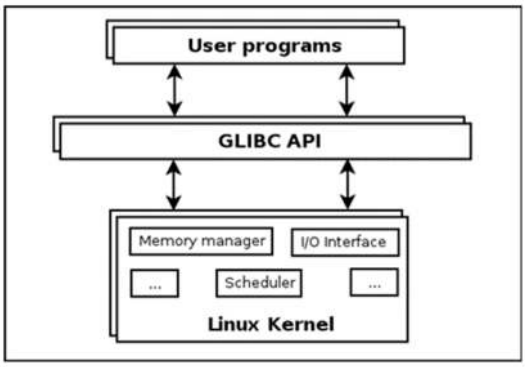


Figure 3. Relationship between the Linux kernel and GLIBC API

To model and implement RTS, the POSIX is commonly used. It offers many advantages, such as portability and standardization of the application development. It is aimed for software compatibility between UNIX-like OSs, such as Linux, defining an API.

**RTS feasibility testing for a CPS**

One important feature of RTS applications is that they must be verified before the execution. It is crucial to know whether the schedule provided by the scheduler is feasible. A schedule is feasible if it respects the deadline of all jobs. An optimal scheduling policy always generates a feasible schedule for a hard RTS, providing that a given set of jobs has feasible schedules [25, 26, 27]. It is important to know that, whether a task set is schedulable or not, i.e., the scheduler always generates a feasible schedule using a specific policy. This problem is commonly known as the feasibility test. [25] proved that the feasibility test in uniprocessor RTSs is a co-NP-complete problem in the strong sense for non-trivial computational models.

There are two types of feasibility tests: exact tests, which check sufficient and necessary conditions, and inexact tests, which check sufficient but not necessary conditions [8]. If a task set is scheduled with a given policy and satisfies the exact test, then all the tasks will be executed according to their deadlines. On the other side, if a task set does not satisfy the inexact test, it is not really known whether all the tasks may complete their execution according to their respective deadlines. Liu and Layland [7] demonstrated that RM is an optimal policy for the static priority assignment. Their inexact test for RM states that a set of *n* periodic tasks is schedulable under RM if:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le n\left(2^{1/n} - 1\right). \qquad (1)$$

The authors showed that the RM policy is able to schedule any periodic task set τ with implicit deadlines (periods are equal to the deadlines) if the total utilization of the processor satisfies $U_\tau \le \ln 2 \approx 0.6931$, where $U_\tau$ is given as follows

$$U_\tau = \sum_{i=1}^{n} \frac{C_i}{T_i}. \qquad (2)$$

Table 2 describes the behavior of this expression for different values of *n*. One can observe in the table that, by increasing the number of tasks, the minimal guaranteed utilization converges to:

$$U_{\min} = \ln 2 \approx 0.6931. \qquad (3)$$

Table 2 – Minimal guaranteed utilization of the processor for *n* tasks

| *n* | $U_{\min}$ |
|---|---|
| 1 | 1 |
| 2 | 0.8284 |
| 3 | 0.7798 |
| 4 | 0.7568 |
| 5 | 0.7433 |
| … | … |
| ∞ | 0.6931 |

A necessary and sufficient (exact) test for the RM policy was proposed by Lehoczky et al. in [28]. It considers the processor utilization by the periodic task set as a function of time at a critical instant. The test is as follows:

Let τ be a set of *n* tasks of the periods $T_1 \le T_2 \le \ldots \le T_n$, respectively, in a uniprocessor RTS. The cumulative demand on the processor by this set of tasks over the time interval $[0, t]$ at a critical instant is:

$$W_i(t) = \sum_{i=1} C_j \left\lceil \frac{t}{T_j} \right\rceil, \qquad (4)$$

where

$$L_i(t) = \frac{W_i(t)}{t},$$

$$L_i = \min_{\{0<t\le T_i\}} L_i(t),$$

$$L = \max_{\{1\le i\le n\}} L_i,$$

$$S_i = \left\{ kT_k \,\middle|\, k = 1, \ldots, \left\lfloor T_i / T_j \right\rfloor; j = 1, \ldots, i \right\}.$$

In this test, $L_i$ is the utilization factor required to meet the deadline of a task $i$, $1 \le i \le n$ over the time range $[0, t]$; $W_i$ is the cumulative demand on the processor by a set of tasks $\tau_1$, …, $\tau_i$, over the time range $[0, t]$; $S_i$ is the set of activation points for a task $\tau_i$. In this manner, a task $\tau_i$ is schedulable under the RM policy if and only if:

$$L = \max_{\{1\le i\le n\}} L_i \le 1. \quad (5)$$

Liu and Layland [7] introduced an exact test of the EDF feasibility for any periodic task set and proved the optimality of the EDF dynamic algorithm for uniprocessor architectures. A periodic task set is schedulable under the EDF policy if and only if:

$$U_\tau \le 1. \quad (6)$$

The modeling of software for hard RTS (i. e., for I40) requires that the designer define the parameters of the system task: deadlines, periods, worst-execution times, among others. Also, a scheduling policy must be selected, and a feasibility tests must be applied, in order to verify that every temporal restriction is satisfied.

## Modeling an RTS

A CPS system is comprised of a computational core (cybernetic system), a communication core, and a controlled system (physical system). The computational core is in essence an RTS. In this section, an example of the modeling of a computational core for a CPS is discussed.

## A controlled system

A prototype of a controlled system was designed to illustrate a CPS system. This system is composed of three threads. It considers a periodic task (thread) that controls the appearance of any external object within a predefined range, and two independent tasks that perform calculations in the background. Such a system can be interpreted as a mobile robot, which checks periodically whether an object makes obstacles in his trajectory. If an object is detected, it turns an LED on immediately. Conversely, when the object is no longer detected, the LED is turned off. These actions are equivalent to the re-adjusting of the robot's trajectory, within a specific timing window, to avoid a collision.

An ultrasonic HC-SR04 was used for the obstacle detection with the goal to detect objects and also to calculate the distance from the sensor to an object in a range [2 ÷ 450] cm. The sensor sends a start pulse and measures the width of the returned pulse. An Arduino UNO microcontroller was used to communicate the ultrasonic sensor with the controlling system (such as, e. g., a computer). Figure 4 shows the architecture of the used RTS prototype. The LED was connected to the GND pin and the pin of the Arduino micro-controller in order to represent the turning ON/OFF of the light. Also, the pins are used for the trigger and echo of the sensor, respectively. Finally, the VCC and GND pins are connected to the 5v and GND pins of the Arduino UNO. When an object is detected within the range, the system checks whether the LED light is turned OFF and turns it ON. If the LED light is turned ON, no action is performed. When the task is activated in its next period and the object is no longer within the detection range, the system checks whether the light is switched ON to turns it OFF. The controlling system communicated with the Arduino micro-controller through the USB serial port, using the Arduino-Serial library [29].

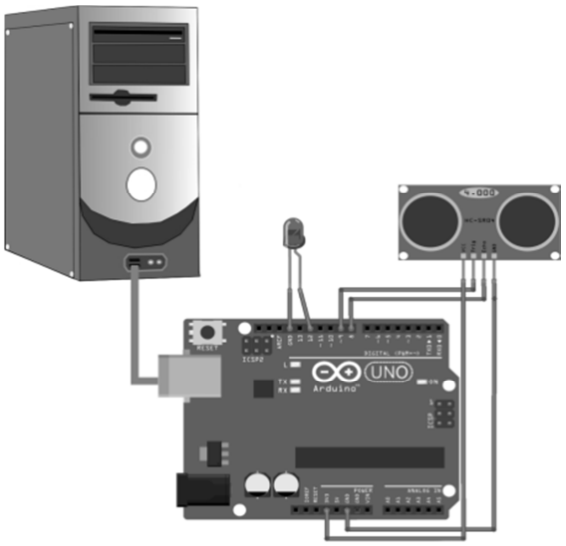

Fig. 4. Architecture of the controlled system

**Experiment**

To evaluate the feasibility of the system, the controlling system was implemented on a Dell Vostro 260 with 6 Gb of RAM and an Intel® CoreTM i3-2100 CPU @ 3.10 GHz. However, since a mono-processor kernel was configured, only one core and 1 Gb of RAM were used. The system ran on a Debian GNU/Linux 7.4.0 Wheezy i386 OS, using the PREEMPT RT 3.4.61edfV2-rt77+ and the SSELINUX-EDF patches. The system used the EDF policy through the SSELINUX-EDF developed by Amaro et al. [23]. The scheduling algorithms used were RM and EDF. The task model was the periodic task model. The system is comprised of three hard real-time tasks. The parameters of the tasks are shown in Table 3.

Table 3 – Parameters of the experimental task set (time is given in ms)

| Thread | $C_i$ | $D_i$ | $T_i$ |
|---|---|---|---|
| 1 | 290 | 700 | 700 |
| 2 | 50 | 600 | 600 |
| 3 | 190 | 400 | 400 |

To model the correctness of the system, a feasibility test must be conducted. The test was carried out using two corresponding applications developed with the aim to simulate realistic scenarios. Three independent threads were created, with different deadlines and WCETs.

Given the time of execution of the tasks, the total utilization of the processor is: $U_\tau = \frac{290}{700} + \frac{50}{600} + \frac{190}{400}$, $U_\tau = 0.972$.

The exact schedulability test of [28] was used to verify the schedulability of the task set using the RM scheduling policy:

$$S_3 = \{400, 600, 700\},$$
$$W_1 = C_1 + C_2 + C_3 = 530 \le 400,$$
$$W_2 = C_1 + C_2 + 2C_3 = 720 \le 600,$$
$$W_2 = C_1 + 2C_2 + 2C_3 = 770 \le 700.$$

The test showed that Thread 1 is not capable to reach the execution time before the corresponding deadline even in the first activation. Threads 2 and 3 showed the same behavior.

Figure 5 displays a Gantt diagram, which shows that the tasks are not schedulable under the RM policy. This means that some deadlines are met, due to the missed deadline by the first job of Thread 1. If the parameters cannot be modified, the tasks would not be executed correctly under the existing scheduling policies in RT-Preempt. However, the value of the total processor utilization $U_\tau \le 1$ implicates that the task set can be still correctly scheduled using the EDF policy.

The next test was performed under the EDF policy as shown in Figure 6. The purple arrows display the activations and deadlines of the threads. The subsequent observations can be derived from the diagram:

- All the task's jobs were fulfilled by the respective deadlines.
- The non-real-time tasks, which were grouped in Idle, were processed only when the real-time tasks did not require to be executed.
- The schedule is similar in both hyperperiods, which are separated by a vertical red line.

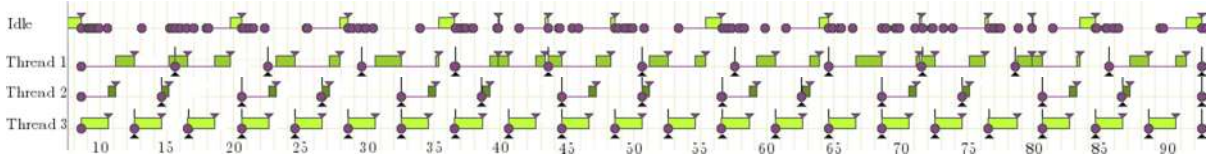


Figure 5. Schedule of the task set using the RM policy

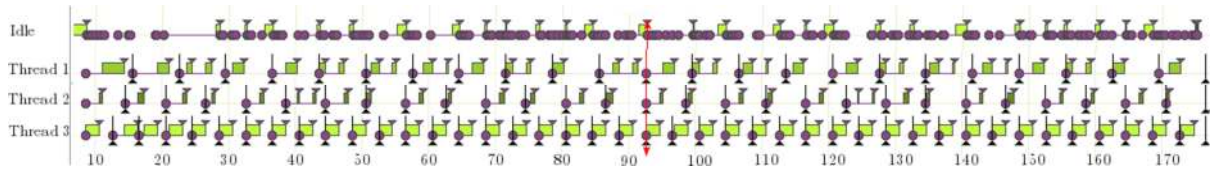


Figure 6. Schedule of the task set using EDF (two hyper periods)

Table 4 displays some metrics obtained for the first hyperperiod. We can draw the following conclusions:

– The required time to switch between two tasks (context switch) with EDF is larger than with RM.

– Each time a task needs to be activated, EDF uses more time than RM.

– There is a double number of preemption points using the RM policy. It corresponds to the observation by [27] that RM introduces more preemptions than EDF.

Nevertheless, despite the FIFO policy requires less time to accomplish context switches, the times of the EDF algorithm are minimized during the execution because it does not produce such a quantity of preemptions.

Table 4 – Metrics of the RM and EDF policies (average time in microseconds)

| Policy | Context switch | Task wakeup | Preemption points |
|---|---|---|---|
| FIFO (RM) | 11.077 | 2.310 | 14 |
| EDF | 12.344 | 3.364 | 7 |

## Conclusion and future work

CPSs are a key component for developing applications for I40. CPSs are comprised of a computational core, a communication core, and a physical system. Since the computational one has strict timing restrictions when model it, the designer must verify that all the temporal restrictions will be satisfied. In this paper, the modeling of the software for the Industry 4.0 was discussed. An example was introduced, showing techniques to determine the correct execution of the system. It was shown that the selection of the scheduling policy of the computational core is fundamental in satisfying the timing restrictions of the system.

As future work, we plan to implement the RTS using an embedded system as the hardware platform.

## References

1. Rajkumar, R., Lee, I., Sha, L., Stankovic, J.. Cyber-Physical Systems: *The Next Computing Revolution*. In proceedings of Design Automation Conference. 731 – 736. 2010.
2. NITRD: Networking and Information Technology Research and Development Program. Cyber-Physical *System (CPS) Vision Statement*. Available on-line: https://www.nitrd.gov/nitrdgroups/index.php?title=File:Cyber_Physical_Systems_(CPS)_Vision_Statement.pdf. 2011.
3. Buttazzo, G. C. (2011). Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer Publisher, 540 p.
4. Davis, R. I., Burns, A (2011). A Survey of Hard Real-time Scheduling for Multiprocessor Systems. *ACM Computing Surveys*, 43(4) 35:1–35:44.
5. Shi, J., Wan, J., Yan, H. and Suo, H. (2011). *A Survey of Cyber-Physical Systems*. In Proceedings of the International Conference on Wireless Communications and Signal Processing. Nanjing. China. 1 – 6. 2011.
6. Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., Baruah, S. (2004). A categorization of real-time multiprocessor scheduling problems and algorithms. Handbook on Scheduling Algorithms, Methods, and Models. Chapman Hall/CRC Publisher, 30 p.
7. Liu, C. L., Layland, J. W. (1973). Scheduling algorithms for multi-programming in a hard-real-time environment. *Journal of the ACM*, 20 (1), 46 – 61.
8. Burns, A. (1991). Scheduling hard real-time systems: a review. *Software Engineering Journal* 6 (3), 16 – 28.
9. Srovnal, V., Kotzian, J. (2008). *Development of a flight control system for an ultralight airplane*. In: Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT 2008), Wisła, Poland, IEEE Computer Society.
10. Faggioli, D., Checconi, F., Trimarchi, M., Scordino, C. (2009). *An EDF scheduling class for the Linux kernel*. In: Proceedings of the 11th Real-Time Linux Workshop, Dresden, Germany, 8 p.
11. Yodaiken, V (1999). *The RTLinux manifesto*. In: Proceedings of the 5th Linux Conference, Raleigh, North Carolina, USA.
12. Koh, J. H., Choi, B. W. (2013). Real-time Performance of Real-time Mechanisms for RTAI and Xenomai in Various Running Conditions.

*International Journal of Control and Automation*, 6 (1), 235 – 246.
13. Calandrino, J. M., Leontyev, H., Block, A., Devi, U. C., et al. (2006). *LITMUS-RT: A testbed for empirically comparing real-time multiprocessor schedulers*. In: Proceedings of the 27th Symposium on Real-Time Systems, Rio de Janeiro, Brazil, 11 – 26.
14. Rostedt, S., Hart, D. V. (2007). *Internals of the RT patch* In: Proceedings of the Linux Symposium, Ottawa, Canada, 161 – 172.
15. Faggioli, D., Trimarchi, M., Checconi, F., Bertogna, M., et al. (2009a). *An implementation of the earliest deadline first algorithm in Linux*. In: Proceedings of the ACM symposium on Applied Computing, Honolulu, USA, 1984 – 1989.
16. Lelli, J., Faggioli, D., Cucinotta, T. (2011). *An efficient and scalable implementation of global EDF in Linux*. In: Proceedings of the 7th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Porto, Portugal, 6 – 15.
17. Dellinger, M., Garyali, P., Ravindran, B. (2011). *CHRONOS Linux: A best-effort real-time multiprocessor linux kernel*. In: 48th Design Automation Conference (DAC), NY, USA, 474 –479.
18. XENOMAI website. http://www.xenomai.org/. (Consulted at October 4, 2018)
19. RTAI website. https://www.rtai.org/. (Consulted at November 4, 2015)
20. VxWORKS website. http://windriver.com/products/vxworks/. (Consulted at October 4, 2018)
21. Ose homepage. http://www.enea.com/ose. (Consulted at October 4, 2018)
22. Marte os homepage. http://marte.unican.es/. (Consulted at October 4, 2018)
23. Amaro, V., Diaz-Ramirez, A., Flores-Rios, B., González-Navarro, F., Werner, F., Burtseva, L. A scheduling extension scheme of the earliest deadline first policy for hard real-time uniprocessor systems integrated on POSIX threads based on Linux. *Computer Systems Science and Engineering*, 33 1), 2018, 31 – 40.
24. GLIBC website. https://www.gnu.org/software/libc/. (Consulted at October 4, 2018)
25. Baruah, S. K., Rosier, L. E., Howell, R. R. (1990). *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*. Real-Time Systems, 2 (4), 301 – 324.
26. Stankovic, J. A., Ramamritham, K., Spuri, M. (1998). Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms. Kluwer Academic Publishers.
27. Buttazzo, G. C. (2005). Rate Monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29 (1), 5 – 26.
28. Lehoczky, J., Sha, L., Ding, Y. (1989). *The Rate Monotonic scheduling algorithm: exact characterization and average case behavior*. In: Proceedings of the Symposium on Real Time Systems, Santa Monica, CA, USA, 166 – 171.
29. Arduino-serial source page. https://github.com/todbot/arduino-serial. (Consulted at October 4, 2018)

**Mayra L. Lizárraga, M.S., Tecnológico Nacional de México/ IT de Mexicali, Av. Tecnológico, S/N, Colonia Plutarco Elías Calles, C.P. 21296, Mexicali, B.C., Mexico, telephon +52 686 580 4980, mayral.lizarraga@gmail.com**

**Andres Buelna, M.S., Tecnológico Nacional de México/ IT de Mexicali, Av. Tecnológico, S/N, Colonia Plutarco Elías Calles, C.P. 21296, Mexicali, B.C., Mexico, telephon +526865804980, andres.buelna.b@gmail.com**

**Arnoldo Díaz-Ramirez, Dr., Tecnológico Nacional de México/ IT de Mexicali, Av. Tecnológico, S/N, Colonia Plutarco Elías Calles, C.P. 21296, Mexicali, B.C., Mexico, telephon +526865804980, adiaz@itmexicali.edu.mx**

**Vidblain Amaro-Ortega, Dr., Tecnológico Nacional de México/ IT de Mexicali, Av. Tecnológico, S/N, Colonia Plutarco Elías Calles, C.P. 21296, Mexicali, B.C., Mexico, telephon +526865804980, vamaro@itmexicali.edu.mx**

**Maryna Volodymyrivna Kostikova, associate professor, cand. eng. sc., Kharkiv National Automobile and Highway University, 25, Yaroslava Mudroho str., Kharkiv, 61002, Ukraine, telephon +380577073774, kmv_topaz@ukr.net**

**Félix Fernando González-Navarro, Dr., Instituto de Ingeniería, Universidad Autónoma de Baja California, Calle de la Normal, S/N, Colonia Insurgentes Este blvd. Benito Juarez, 21280, Mexicali, B.C., Mexico, telephon +526865664150, fernando.gonzalez@uabc.edu.mx**

**Frank Werner, Prof., Dr., Institute of Mathematical Optimization, Otto-von-Guericke University, Universitätsplatz 2 39106, Magdeburg, Germany, telephon +493916752025, frank.werner@ovgu.de**

**Larysa Burtseva, Dr., Instituto de Ingeniería, Universidad Autónoma de Baja California,**

**Calle de la Normal, S/N, Colonia Insurgentes Este blvd. Benito Juarez, 21280, Mexicali, B.C., Mexico, telephon +526865664150, burtseva@uabc.edu.mx**

## АСПЕКТИ ТИМЧАСОВИХ ОБМЕЖЕНЬ ПРИ ВИМІРЮВАННІ КІБЕРФІЗІЧЕСКІХ СИСТЕМ В ПРОМИСЛОВОСТІ

***Анотація.*** *Кіберфізичні системи (CPS) в Індустрії 4.0 складаються з обчислювального ядра, ядра зв'язку й фізичної системи. Обчислювальне ядро має строгі часові обмеження і являє собою систему реального часу (RTS). RTS, реалізовані в традиційному встаткуванні, використовуються в широкому спектрі додатків, таких як пристрої загального користування, відомі як Internet of Things (Iot), а також пристрої зв'язку, мобільні телефони або комп'ютери. Ці системи викликають великий інтерес через тенденцію до автоматизації систем і додатків для щоденного використання, наприклад, самокеровані автомобілі, автономні літаки, датчики й роботи для догляду за людьми похилого віку. Одна з основних особливостей RTS полягає в тому, що вони мають строгі часові обмеження, які повинні бути виконані, оскільки в протилежному випадку результати можуть бути катастрофічними. RTS виконуються в операційній системі реального часу (RTOS). Однієї з найбільш популярних операційних систем є Linux. У якості алгоритмів планування в Linux використовуються політики планування на основі пріоритетів First-In First-Out (FIFO) і Round Robin (RR), але вони недостатні для жорстких RTS. У цій статті пропонується інтегрування політик Rate Monotonic (RM) і Erliest Deadline First (EDF) у систему Linux для планування періодичних завдань у жорстких RTS. Тест на виконуваність розкладів в однопроцесорних RTS є co-NP-повной задачею у суворому смислу для нетривіальних обчислювальних моделей. Лехоцким був запропонований необхідний і достатній (точний) тест для політики RM. Він розглядає використання процесора набором періодичних задач, як функцію часу в критичний момент. Лю й Лейленд увели точний тест виконуваності EDF для будь-якого періодичного набору завдань і довели оптимальність його динамічного варіанта для однопроцесорних архітектур. Для ілюстрації CPS був розроблений прототип контролюючої системи, що полягає із трьох потоків (threads). Виконуваність розкладів отриманих з використанням EDF у RM була перевірено за допомогою обох тестів. Тести показали, що використовуючи політикові RM, один із трьох потоків не зміг виконатися до відповідного крайнього строку при першій же активації. Однак значення загального використання процесора показало, що набір задач може бути правильно запланований з використанням EDF. При цьому всі завдання виконувалися відповідно до встановлених строків; завдання не потребуючі виконання в реальному часі, були згруповані в незайнятому (Idle) часі й оброблялися тільки тоді, коли задачі реального часу не вимагали виконання; отриманий розклад аналогічний у двох досліджених гіперперіодах.*

***Ключові слова:*** *система реального часу, кіберфізична система, планування, Індустрія 4.0, алгоритм Earliest Deadline First, алгоритм Rate Monotonic.*

## АСПЕКТЫ ВРЕМЕННЫХ ОГРАНИЧЕНИЙ ПРИ ИЗМЕРЕНИИ КИБЕРФИЗИЧЕСКИХ СИСТЕМ В ПРОМЫШЛЕННОСТИ

***Аннотация.*** *Киберфизическая система для Индустрии 4.0 состоит из компьютерных программ, коммуникационных программ и физической системы. Компьютерные программы имеют строгие временные ограничения и представляют собой системы реального времени. В этой статье дискутируются вопросы моделирования таких программ.*

***Key words:*** *система реального времени, киберфизическая система, планирование, Индустрия 4.0, алгоритм Earliest Deadline First, алгоритм Rate Monotonic.*