EDF scheduler technique for wireless sensors networks: case study

Rym CHÉOUR[1] , Sébastien BILAVARN[2] , Mohamed ABID[1]
[1] CESlab, National School of Engineers of Sfax, Sfax, Tunisia
[2] LEAT, University of Nice-Sophia Antipolis, CNRS, Nice, France
rym.cheour@ceslab.org, mohamed.abid@ceslab.org
Sebastien.BILAVARN@unice.fr

Abstract

Today, thanks to the recent advances in wireless technology, new products using wireless sensor networks are employed. However, despite the excitement surrounding the wireless sensor networks, its entry into force, is not immune to the problem of energy consumption. To overcome this deficiency and to enhance the real time aspect, a growing interest lies in the implementation of an "Earliest Deadline First" (EDF) scheduler. Thus, we will establish a management policy of periodic tasks that is preemptive, multiprocessor and dynamic. Our target is to implement a real-time scheduling policy as a part of a user-level threads package under the Linux operating system since Linux does not support EDF. Furthermore, this paper describes the technique of the EDF scheduler and how it can yield to significant power savings.

Key words: real time scheduling, EDF, WSN, energy consumption, Linux scheduler

1 Introduction

The network technologies of wireless sensor have become a global trend in communication, mobility and research of flexible implementation. With these advantages, these networks are undoubtedly among the principal vectors of development of embedded real time system[1]. Because they control or monitor real time processes, they must be able to respond to requests within a certain time limit. Confined essentially to autonomous applications and small networks where man could hardly intervenes, the energy source appears to be, therefore, the highest priority in the design and development of sensor networks[2]. In fact, it poses several challenges that the real-time scheduling seems to take up[3]. In the other hand, sensors networks are commonly used in environments where the guarantee of the response time is vital. The system must be flexible enough to get used to a dynamic and changing environment able to meet its deadlines and to detect temporal conflicts, caused by different resources. Besides, meeting temporal deadlines leads to many problems that real-time scheduling can solve [3]. Once confined to a limited role and thanks to its impact on minimizing the consumption of energy, especially for sensor networks the scheduling is now a basic entity of the development of real time systems.

A scheduling algorithm is perceived as a set of rules that select the task to run at any time during the life of a system [4]. Therefore, we can consider scheduling as an algorithm that allocates the basic units of time called time quantum. A strict real-time system is essential to ensure the respect of deadlines for each task. The deadlines consist of run-ability constraints (each task must be completed before the next request) [4]. Thus, a scheduling policy is applied to check the deadline of each task, the material constraints and the dependencies among data.

In this paper, the EDF scheduler aims at evaluating a task set with given properties in terms of schedulability and compliance with given execution time constraints. It exactly consists in implementing and estimating such policy in an operating system, such as the Linux.

The remainder of this paper is organized as follows. In the first section, we introduce Linux's most important abstraction, the process or the task model for basic process management including the scheduling [5]. Then, we discuss an issue related to the specific policies of energy's management in the sensors networks. The following section deals with the fundamentals of the EDF. Next, we will give a concise overview of the Linux process scheduler, its scheduling algorithm and its API. Furthermore, we outline the experimentation and the results observed within the scheduler. Also, we compare the performances of our scheduler developed on Linux with the results obtained with the simulation multiprocessor scheduling tool STORM [6].

2 State of art

The majority of scheduling strategies uses the concept of task. Several models of recurring real-time tasks have been defined. Belonging to one of these families influences strongly how the system will operate and particularly the type of the algorithm to use. We will try to give a glance about the task models and an overview related to the different technique to reduce energy consumption.

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille

2.1 Tasks models

Tasks can be grouped into three families: periodic, aperiodic and sporadic. The simplest and the most fundamental model is provided by the periodic task model of Liu and Layland [4]. The periodic tasks are those whose processing is repeated on a regular basis such as the regular monitoring of the state of a physical sensor or sampling of the serial communication line.

T_i a periodic task is characterized by the quadruplet (O_i, T_i, D_i, C_i)[7], where:

- The date of arrival O_i, is the moment of the first activation of the task τ_i
- Time of execution C_i specifies an upper limit on the time of execution of each task τ_i.
- The relative deadline D_i denotes the separation between the arrival of the task and the deadline (a task that arrives at time t has a deadline at $t+D_i$);
- A period T_i denoting the duration between two successive activations of the same task.

2.2 Optimization of energy consumption

The wireless sensor must be fitted with a battery-powered covering several years and offering total energy independence. Notwithstanding, the battery technology is not progressing fast enough to satisfy their requirements [8]. Different solutions are possible to minimize the energy consumption of WSN. Autonomous power supplies can be well designed to capture tiny amounts of energy from their environment. Even when available, energy-efficient products become essential to reduce thermal losses evacuated by expensive means of cooling which are responsible for failure. This approach requires low-power efficient components of energy. Seeming trivial, the process is often complex. The first parameter we mention is the consumption in normal times, of the processor the sensor, the radio transceiver and others components such as external memory and peripherals [2].

2.2.1 Management of static power consumption

Many methods can reduce the activity of these circuits such as clock gating or management of low-power modes. The clock gating can cut part of the clock tree to avoid switching of unused parts of the circuit [2]. However, it is impossible to control all the unnecessary commutations [8]. But, if the scheduler is not suitable, the significant energy savings are achieved at the expense of the system responsiveness. Indeed, stopping and restarting clocks cause latencies and increase consumption. The difficulty is to know what should be done to avoid compromising the processing of an outside event while minimizing the amount of energy expended. The use of these methods is often optimal. Therefore, it is necessary to use a scheduler which includes tasks' execution wherever possible and in return has long periods of inactivity[2].

2.2.2 Dynamic power consumption

QDI "Quasi Delay Insensitive" circuits are a class of almost delay insensitive asynchronous circuits which are invariant to the delays of any of the circuit's elements [9] [10]. The synchronization between the blocks is done locally by requests /acquittals. So, only the parts of the circuit making a calculation have an activity. The rest of the circuit consumes very little energy and wakes up immediately when it is requested. This decreases the consumption and reduces the dynamic consumption significantly. This particular property is exploited to manage the levels of tension circuit DVS "Dynamic Voltage Scaling" effectively. Indeed, the dynamic adjustment of voltage (DVS) is a very important technique to reduce energy consumption [11]. However, sensor networks must manage these tensions at a lower cost. This method saves energy at around 45%[3].

2.2.3 Impact of the scheduling policy

Using tasks scheduling, we try to combine the minimization of the consumption of a WSN node and to ensure a maximum of performance to users. In addition, the strategies of scheduling reduce the consumption of energy considerably while they reduce also the frequency of the processor [12].

It is possible to optimize the lifetime of the network at different levels. As a node has a very low activity within the network, it is desirable from the standpoint of consumption, and therefore the lifetime of the network, to reduce the electrical activity of the circuits, particularly in periods of inactivity [2]. Thus, it is necessary to characterize the activity of the wireless sensors network in terms of maximum number of instructions and deadlines so as to schedule them and to calculate the minimal speed of the processor required to comply with time constraints. As this speed increases considerably due to the intense solicitation of multiple tasks per processor, we notice that simultanously, energy consumption increases.

2.3 Earliest Deadline First (EDF)

The algorithm "Earliest Deadline First" (EDF) [4] is a preemptive real time and it uses a dynamic priority scheduling algorithm. It assigns priority to each task depending on the deadline. As the deadline of a task is closer, its priority is higher. In this way, the more quickly the work must be done, the more chance it has to be executed. This algorithm is proved to be optimal in the sense that if a system of tasks can be sequenced using any policy of assigning priorities, the system can also be sequenced with the EDF algorithm [13]. The

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille

study of schedulability gives a necessary and sufficient condition formulated by the following theorem: a system of periodic tasks can be sequenced using the EDF algorithm if and only if:

$$\sum_{i=1}^{n} \frac{Ci}{Ti} \leq 1 \quad (1)$$

Moreover, the ins and outs of this scheduler represent its ability to ensure a maximum occupancy of the CPU up to an upper limit of 100% CPU utilization [14].

The EDF scheduler combined with an algorithm of tension management "DVFS" Dynamic Voltage and Frequency Scaling can calculate the voltage applied to the processor and subsequently adapt it to the parameters of each task[15]. Knowing the worst case execution of the task, we can predict that the next invocation will not exceed the deadline. Furthermore, we can take advantage of the idle time tasks to reduce the speed. Thus, a small decrease of the tension slows the circuit slightly, but it can reduce the energy consumption significantly (the energy E is proportional to the square of the voltage). Moreover, it is also possible to vary dynamically the voltage of a circuit depending on its activity to reduce consumption [10]. Since, the good management of processes governing the sensor network is proved to be necessary, even crucial.

3 Proposed technique for the processes management

The design process for a real-time application involves splitting the application code into tasks. A task, also called a thread, is an infinite loop that has its own stack area, its own set of CPU registers, its own purpose and a priority assigned based on its importance. A running Linux application is composed of one or more tasks. The kernel of a Linux system is essential to execute and to enable them to interact[15].

3.1 Multithreaded programming

Multitasking or multithreading is the process of scheduling and switching the CPU (Central Processing Unit) between several tasks; a single CPU switches its attention between several sequential tasks. Multithreaded programming is the art of programming with threads. The most common API on Linux for programming with threads is the API standardized by IEEE Std 1003.1c-1995 (POSIX 1995 or POSIX.1c). Developers often call the library that implements this API pthreads[5][16] [17].

3.2 States of tasks

Hence, as the multitasking system runs, each task exists in one of these four states: running, ready for execution, waiting or terminated as shown in figure 1. The transition from one state to another is done through system calls or a decision of the scheduler. When a multitasking kernel decides to move the running task to another state and to give control of the CPU to a new task, a context switch should be performed [12].

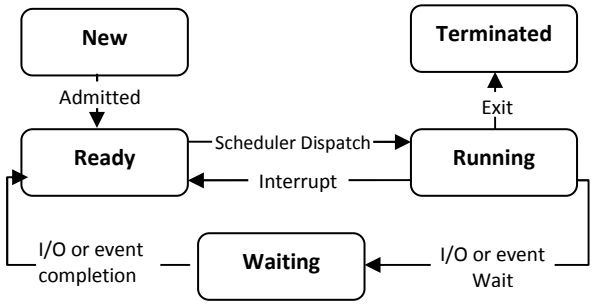

Figure 1: State of a task.

A new released task is ready when it can execute but its priority is less than the currently running task. A task is running when it has control of the CPU. A task is waiting for an event when it requires the occurrence of an event. Finally, a task is interrupted when an interrupt has occurred and the CPU is in the process of servicing that interrupt.

3.3 Case study Linux: scheduler

Unfortunately, Linux is not in fact a real-time system. Indeed, the Linux kernel is based on the concept of timeshare and not real time. Several technical solutions are already available to improve the behavior of the kernel to make it compatible with the constraints of a real time system [5] [17] [18]. In this respect, the technical solutions available are divided into two categories:

1. The patch called "preemptive" to improve the behavior of the Linux kernel by reducing its latency. Those changes do not transform Linux kernel into a hard real time system. Yet, we can obtain satisfactory results in the case of soft real time constraints.

2. The real time auxiliary kernel believing that the Linux kernel is not really a real time: developers of this technology add to this core a true “real-time scheduler” with fixed priorities. This auxiliary core addresses real-time tasks directly and delegates other tasks to the Linux kernel, being a lower priority task. This technique allows the introduction of hard real-time systems.

3.4 Scheduling policies under Linux

The scheduler is the part of a kernel that decides which runnable process will be executed next by the CPU. The Linux scheduler offers three different scheduling policies, two for real-time applications and one for other processes. A preprocessor macro from the header <sched.h> represents each policy: the macros are SCHED_FIFO, SCHED_RR, and SCHED_OTHER defined in the standard POSIX.b. SCHED_OTHER (default) which is a

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille

scheduling time-shared tasks and which also is used by most processes. SCHED_FIFO and SCHED_RR are provided for real-time applications that require precise control of the selection process [5][17],[18]. A static priority value sched_priority is assigned to each process and this value can be changed only via system calls. For normal applications, this priority is always 0. For the real-time processes, it ranges from 1 to 99, inclusive. The Linux scheduler always selects the highest-priority process to run.

3.5 Processor affinity

Processor affinity refers to the tendency of a process to get scheduled constantly on the same processor. As Linux supports multiple processors in a single system, the scheduler must ensure full use of the system's processors, because it is inefficient for one CPU to sit idle while a process is waiting to run [5]. On a symmetric multiprocessing (SMP) machine, the process scheduler must decide which processes run on each CPU. SMP lets multiple CPUs share the same board, memory, I/O and operating system. Nevertheless, each CPU in a SMP system can act independently. Due to the design of modern SMP systems, the caches associated with each processor are separate and distinct.

4 Experimentation

The principle of the EDF policy is to execute the tasks according to their urgency [2]. In contrast, the unavailability of EDF on Linux is not necessarily prohibitive for its use. Certainly, it is possible to implement EDF in the application level as a "leader" task able to schedule the activities of the system. We apply the SCHED-FIFO algorithm to the first N tasks ready to be executed. The kernel places all runnable processes on a ready list. Once a process has exhausted its timeslice, it is removed from this list. EDF can assign a dynamic priority to these tasks in the queue. The end of the execution of a task or its new arrival in the system leads the scheduler to select among all tasks ready to run one whose deadline is the closest. Moreover, the algorithm looks for the shortest deadline in each invocation of the scheduler. In this case, this task is provided with the highest priority. It will be executed immediately and it will be allocated to the available processor. Besides, priorities are assigned on dynamic parameters. However, a task can be accomplished only if all tasks which have smaller deadlines completed their execution or are not active yet. The notion of periodicity in Linux doesn't. So, the development of our scheduler we ought to introduce this concept.

The scheduler must be preconceived intelligibly and should be portable and adequate to time constraints. Therefore, this work aims at implementing an architecture formed by different modules:

- A module "application" representing the thread in question.
- A module "scheduler" governing the functioning of the EDF algorithm.
- A module "utilities" that contains the basic functions of the scheduler.

4.1 Application

More and more applications take advantage of the high performance of threads. It maximizes the utilization of the CPU, increases the speed of the response time and improves the structure, efficiency and design of our scheduler. On a multiprocessor system, each thread can be executed on one processor increasing then the speed of execution significantly. A thread has a data structure which contains the characteristics of the task. It is shared by all tasks and is used by the operating system especially by the scheduler for the arbitration of the needs and the resource demands. Therefore, it allows, for example, to evaluate the behavior of the system when it exceeds time constraints. The runtime behavior of a task does not depend on the others. The table 1 shows the attributes of the threads with a hardware architecture composed of 2 processors and a software architecture composed of 4 periodic independent tasks.

Table 1: Example Task Set

	T0	T1	T2	T3
WCET	8	7	10	9
PERIOD	20	15	25	14
DEADLINE	0	0	0	0

We assume that the deadline is equal to the period.

4.2 Utilities

Linux implements its own interfaces to handle time features. It includes setting and retrieving the current time, calculating elapsed time, sleeping for a given amount of time, performing high-precision measurements of time, and controlling timers[5] This phase covers the data structures representing the time-related chores. It provides extreme flexibility in terms of the time management and also in the assignment of the available processors to the ready tasks according to Linux settings. Otherwise, the operating processor may include periods of inactivity that leads to unnecessary waste of energy. To maximize the performance and the efficiency of the scheduler, we use this idle time to run other high priority tasks. Alternatively, we actuate those processes to sleep, and awake them only when needed, freeing the processor for other tasks.

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille

4.3 The scheduler

The kernel provides a mechanism to ensure a multitasking behavior [3]. This guarantees the equitable distribution of the access to CPUs by the various tasks. A process may need the CPU for example, for calculations, for triggering an interruption, etc. Most hardware components, especially the CPU of a computer are not able to perform multiple treatments simultaneously. The choice of the next "Running" task is the responsibility of the scheduler. A good implementation of the scheduler can trim a few microseconds to process and claim a high accuracy. The figure 2 illustrates an example of execution that follows the next steps:

- Several tasks become ready to run
- The threads are queued according to their priorities in the ready list
- If there are more ready threads to run than CPUs, the operating system scheduler will use thread priority to decide which one runs first.
- Multiple threads run simultaneously per threads

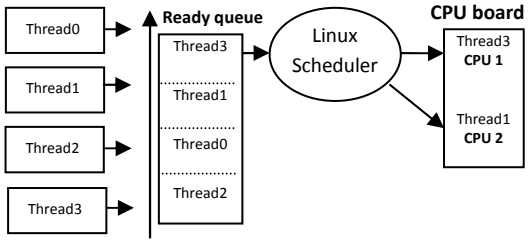

Figure 2: Example of execution

5 Results and test of the EDF scheduler

Achieving the EDF scheduler was initially preceded by the implementation of a test application in C language based on some POSIX threads each one incorporating a WCET "Worst Case Execution Time". Moreover, this setting allowed us to evaluate the performance of the Linux scheduler. It is important to note that the actual sequence of the task's execution obtained with the Linux scheduler was not the expected one. This affects the functioning of the system and slows down its performance.

5.1 *Results*

We try to monitor the status of execution and to respond to any change of state due to a defect (improper shutdown of a task)and by crossing a threshold (exceeding the period…). We have managed to establish a scheduling policy that is dynamic, preemptive and especially accurate. Certainly, the various used- time-parameters were essential in identifying the origin of these operational hazards. When slow tasks are run, a one-second delay between the command action and its execution is acceptable, but meeting the strict time constraints is necessary. Indeed, a high solicitation could affect the real-time capabilities of the system that may not respond within the time limit any longer. Taking into account time constraints, which is as important as the accuracy of the results, entails not only to deliver accurate results, but also to meet the deadlines.

5.2 Result verification

As a matter of fact, an aspect of parallelism appears during the execution and offers a high level of quality and reliability. We assign to the first processor the first two tasks and to the second one the others tasks. The obtained results in terms of priorities and respect of the principle of processor affinity confirm the choice of our scheduling algorithm. Equipped with a dynamic priority and if a high priority task occurs it takes the place of another with lower priority. When a task ends and the field "onTerminated" is set to 1, the ready list varies involving a new sequence in accordance with the foundations of the EDF algorithm.

To control and to ensure the reliability of each task, we get the PID, the priority, its start and the end time. Without changing any parameter of the scheduler, it should be noted that the results are consistent from one simulation to another. This is done due to the determinism of the Linux system.

5.3 Simulation with STORM

To simulate and evaluate the performances of the scheduler, we have adopted the simulation tool STORM which stands for "Simulation TOol for Real time Multiprocessor scheduling". This simulator is able to take into account the requirements of tasks, the characteristics and functioning conditions of hardware components and the scheduling rules. Depending on the scheduling policy and the resources described via an XML file, it runs every task over a specified time interval [4]. The results of the simulation return a set of diagrams as following in figure 3.All these diagrams allow us to analyze the behavior of the system (tasks, processors, timing, performances ...). A window shows a Gantt diagram of every task over an interval from 0 and ends to date 50 (default values). The title of the window refers to the name given to the task in the XML file (PTASKT1, PTASKT2…). On the other hand, it allows us to observe the assignment of the task to the processors through the CPUA and CPUB diagrams over the same interval. The allocation of processors is established according to their availability and to the priority of the task. The preemption is also supported by this simulator. That feature reduces the latency of the system when reacting to real-time or interactive events by allowing a low priority

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille

process to be preempted even if it is in kernel mode executing a system call. This allows applications to run more reliably even when the system is under load.

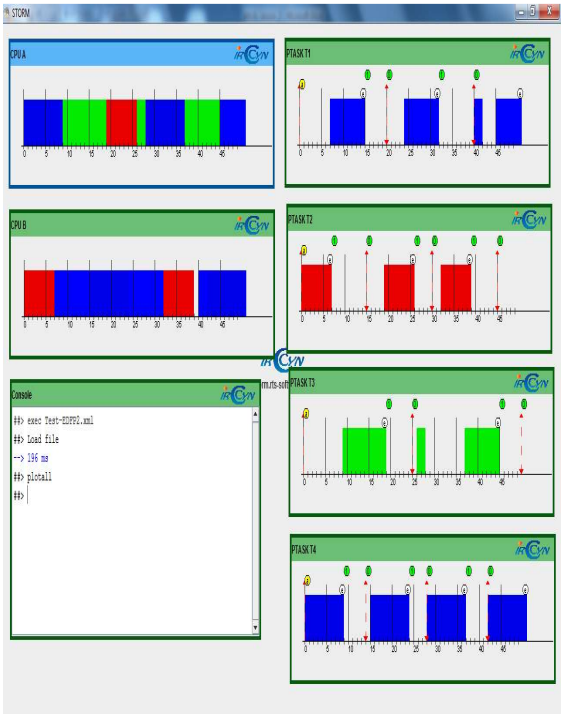

Figure 3: Results with STORM

However, we note that some tasks switch from one CPU to another unlike their execution with Linux such as the task 3. Therefore, we should draw attention to the fact that this simulation engine relies on the priority of the tasks more than on the processor affinity.

6 Conclusion

Time management and task scheduling are required to enhance performance and to improve predictability of the wireless sensor networks. This requirement has led to the wide availability of operating systems to ensure execution schedules where deadlines are met. We have elucidated in this paper the various steps taken for the specification, the development and the impact of the implementation of the EDF scheduler under the Linux operating system. The biggest reason for using Linux is to provide responsiveness to real-time events; it guarantees a deterministic and an optimum task-level response. In addition, it streamlines applications development in complex systems. On the other hand, we have compared the performances with the simulation multiprocessor scheduling tool STORM. This scheduler has a lot of advantages. It is more reliable through the use of preemption, and easier to handle thanks to the dynamic tasks management. A futuristic approach would be to consider a dynamic algorithm, which applies a couple of voltage and speed to the processor depending of the features of every task in the system. It can then be extended to take into account energy configurations like the DVFS "Dynamic Voltage Frequency Scaling", or the DPM "Dynamic Power Management".

7 References

[1] David Culler, Deborah Estrin and Mani Srivastava. "*Overview of Sensor Networks*". In IEEE Computer, vol. 37, no. 8, pp 41–49, august 2004.

[2] Aurélien Buhrig, Marc Renaudin, « *Gestion de la consommation des noeuds de réseau de capteurs sans fil* », Colloque national du GDR SOC-SIP, 2007.

[3] David Decotigny. « *Bibliographie d'introduction à l'ordonnancement dans les systems informatiques temps-réel* ». Technical report, INSA Rennes, 2002

[4] C. L. Liu and J. W. Layland. "*Scheduling algorithms for multiprogramming in a hard-real-time environment*". *ACM*, 20(1):46-61, January 1973

[5] Robert Love, "*Linux System Programming*", O¨Reilly Media, Septembre 2007.

[6] *http://storm.rts-software.org/doku. php*

[7] A. Burns and A. J. Wellings. "*RealTime Systems and Programming Languages"*. Addison Wesley Longman, 4th edition, 2009.

[8] Aurélien Buhrig « *optimisation de la consommation des nœuds de réseaux de capteurs sans fil* »,Thèse, Institut National Polytechnique De Grenoble, Avril 2008.

[9] K. Van BERKEL. "*Beware the isochronic fork. Integration*", the VLSI journal, 13(2) : 103–128, 1992.

[10] A.J. MARTIN. "*The limitations to delay-insensitivity in asynchronous circuits*". In William J. Dally, editor, Advanced Research in VLSI, pages 263–278. MIT Press, 1990.

[11] F. Bouesse, M. Renaudin, A. Witon, F. Germain, "*A Clock-less low-voltage AES crypto-processor*", European Solid-State Circuits Conference (ESSCIRC 2005), Grenoble, France, September, 12th – 16th, 2005, pp. 403-406.

[12] Ahmed RAHNI « *Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et EDF*» Thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique, 05 december 2008.

[13] N. C. Audsley, A. Burns, M. F. Richardson, and A. J.Wellings. "*Hard Real-Time Scheduling: The Deadline Monotonic Approach*". In Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software, Atalanta, 1991.

[14] J. A. Stankovic and M. Spuri and K. Ramamritham and G. Buttazzo, "*Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*", Kluwer Academic Publishers, 0-7923- 8269-2,1998

[15] Pouwelse J., Langendoen K., Sips H., « *Dynamic voltage scaling on a lowpower microprocessor* », Proceedings of the 7th annual international conference on Mobile computing and networking (MobiCom'01), New York, NY, USA, ACM Press, p. 251–259, 2001.

[16] B. Nichols and D. Buttlar and J.P. Farrell, 1996, "*PThreads programming*", O'Reilly, 1-56592-115-1.

[17] T. Ungerer, B. Robic, and J. Silc. "*Mutithreaded Processors*". The Computer Journal, 45(3) : 320–348, 2002.

[18] W. Richard Stevens, Stephen A. Rago, " *Advanced Programming in the UNIX Environment: Second Edition*", Addison Wesley Professional, ISBN: 0201433079, Pages: 960, June 17, 2005

ISBN 978-0-473-16942-8

Editors: S.C.Mukhopadhyay, A. Fuchs, G. Sen Gupta and A. Lay-Ekuakille