Exploitation of the EDF scheduling in the wireless sensors networks

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 sensors 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 systems[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 supply 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 cope with a dynamic and changing environment and to be 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, scheduling is now a basic entity of the development of real time systems. Currently, the use of multiprocessor solutions for sensor networks is not obvious. However, as perspective in our work we aim to manage complex applications (video processing and others). 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 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 tool for multiprocessor scheduling 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 of different techniques that reduce energy consumption.

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 which 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 of the time of execution of each task τ_i.

- The relative deadline Di 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 Power consumption

The CMOS (Complementary Metal Oxide Semiconductor) is the dominant technology in electronic circuit. So, the power consumed is divided into two parts which are the static and the dynamic power. In CMOS, the dynamic power presents 80-85 % of the whole consumption [3]. Most of the times, we consider that the static power of the core is negligible [9]. In this condition the total power consumption is as done in the following equation:

$$P = \alpha C V^2 F \qquad (1)$$

Therefore, the dynamic power consumption is a function of C which is the total capacitance of all the circuits that need to be charged during signal transitions. V is the supply voltage applied to the devices, F is the operating frequency and α is the switching activity.

2.3 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].

2.3.1 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 parts 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].

2.3.2 Static 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 [10] [11]. 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 voltage circuit DVS "Dynamic Voltage Scaling" effectively. Indeed, the dynamic adjustment of voltage (DVS) is a very important technique to reduce energy consumption [12].

2.3.3 Dynamic Power management

Most of microprocessor systems are characterized by a variable amount of calculations in time. Thus, the Dynamic Power Management (DPM) is a technique that reduces the consumption on the system level which, judiciously and selectively, decides to place certain parts of the system in modes (or states) known as low consumption. This comes from the finding that the systems are often conceived and dimensioned for loads and performances peaks which are far from being reached most of the time. While cutting off the

supply or the clock dynamically of the unused resources of the system, the DPM exploits the phases of rest to decrease the static consumption in the system. If the energy and performance overheads in sleep-state transition were negligible, then a simple greedy algorithm that makes the system enters the deepest sleep state when idling would be perfect [13]. The DPM, in general, is not a trivial problem. Indeed, the cost of transitions between the states is a little expensive from the energy point of view [14].

2.3.4 Dynamic voltage and frequency scaling

The appearance of variable voltage processors has led to greater autonomy and energy savings. Dynamic voltage and frequency scaling (DVFS) is an effective technique for reducing CPU energy. The DVFS tries to combine the performance and the lifetime of the battery. A number of modern microprocessors such as Intel's XScale and Transmeta's Cruso are equipped with the DVFS functionality[3]. The first feature of this technique provides high performance only for a short time reduced, while the rest of the time a low CPU power is largely sufficient [13] [14]. Most micro-processor systems are characterized by a time-varying computational load. DVFS exploits the CMOS property that a linear reduction in the supply voltage results in a cubic reduction in the power consumption at the expense of a linear slow down in the processor frequency [12]. It is better thus to run the processor at the weakest frequency compatible with the necessary performance level. When used at a reduced frequency, the processor can operate at a lower supply voltage.

As wireless sensor networks interfere in a growing number of applications ranging from simple environmental monitoring like temperature detection to complex calculation such as video processing. This last type of application requires a high load at the sensor level and leads to a problem of optimization. It is in this perspective that EDF DVFS technique is justifies itself.

2.4 Impact of the scheduling policy

By using tasks scheduling, we try to combine the minimization of the consumption of the processor 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 [15]. 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 simultaneously, energy consumption increases.

2.5 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 [16]. The 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:

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i}\right) \leq m \quad (2)$$

m represents the number of the processor.

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 [17].

The EDF scheduler combined with an algorithm of voltage and frequency management "DVFS" (Dynamic Voltage and Frequency Scaling) can calculate the frequency applied to the processor and subsequently adapt it to the parameters of each task [18]. 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 frequency slows the circuit slightly, but it can reduce the energy consumption significantly. Moreover, it is also possible to vary dynamically the voltage and the frequency of a circuit depending on its activity to reduce consumption [11]. Since, the good management of processes governing the sensor network is proved to be necessary, even crucial.

2.6 EDF scheduler in TinyOS

An interesting way to reduce consumption in sensor networks was originally proposed with TinyOS developed at Berkeley University [19]. TinyOS is an event-driven operating system that is confined to specific applications of wireless sensors networks

[19]. It does not have space kernel-user and does not allow dynamic allocation of tasks. In addition, the mechanism of preemption between tasks is absent. What may cause various problems such as starvation and monopolizing the processor at the expense of other tasks. Similarly, it should also be noted that the scheduler is fixed and it is impossible to modify. A non-preemptive EDF algorithm has been implemented with TinyOS. However its complexity and its static and non-preemptive property have limited its performance. Therefore, Linux seems to suit better the implementation of the EDF scheduler.

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 assignment 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 the tasks and to let them interact[20]. As the kernel has always the highest priority, it is necessary to pay attention to the response times of the scheduler which is located in user space. The reason to develop in user space is that it is much easier than in kernel space and it will allow us to evaluate the energy profits by measuring consumption on a board.

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][20] [21].

3.2 States of tasks

The proposed technique provides various task types. Hence, as the multitasking system runs, we assign for each task one of these four states: running, ready for execution, waiting or terminated as shown in figure1. The transition from one state to another is done through system calls or a decision made by 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 [15].

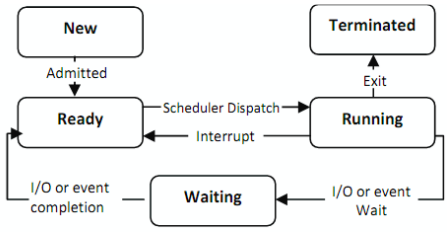

Figure 1: States of a task.

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] [20] [21]. Respectively, 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 one: 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 new 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][20][21]. 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. 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' 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 policy to the first N tasks ready to be executed. The scheduler places all runnable processes on a ready list. Once a process has exhausted its time slice, 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 exist. So, in 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 packages:

• A package called "application" representing the threads in question.

• A package called "scheduler" governing the functioning of the EDF algorithm.

• A package called "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 significantly the speed of execution. A thread has a data structure which contains the characteristics of the task. It is shared by all tasks and is used especially by the scheduler for the arbitration of the needs and the resource demands. Therefore, the parameters of each thread allow, for example, supervising the behavior of the system. The runtime behavior of a task does not depend on the others. Table 1 shows the attributes of the threads with a hardware architecture composed of two 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 functions 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 cores. It provides extreme flexibility in terms of 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. The synchronization mechanism used here to suspend and resume task executions is based on pthread conditions.

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 should not excess a few microseconds to process and provide low response time.

Figure 2 illustrates an example of execution that follows these 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 scheduler will use thread priorities to decide which one runs first.

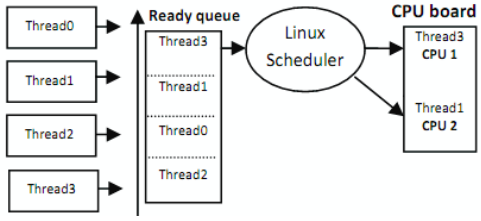

Figure 2: Example of execution

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. We take an example task set, composed of four periodic tasks whose parameters are shown in Table 1. Those tasks are independent and are assigned to a platform made of two processors. We have defined a specific structure of task which contains the necessary information for the scheduler such as the “actual execution time” (aet), the state of the task (ready, waiting…), the next deadline and incorporating a WCET “Worst Case Execution Time”, a period and a deadline etc, as shown in figure 3. Thus, we have considered the same task set as it is done with STORM. Moreover, this setting allowed us to evaluate the correctness and the performance of the user space scheduler. It is important to note that a false sequence of execution affects the functioning of the system and slows down its performance.

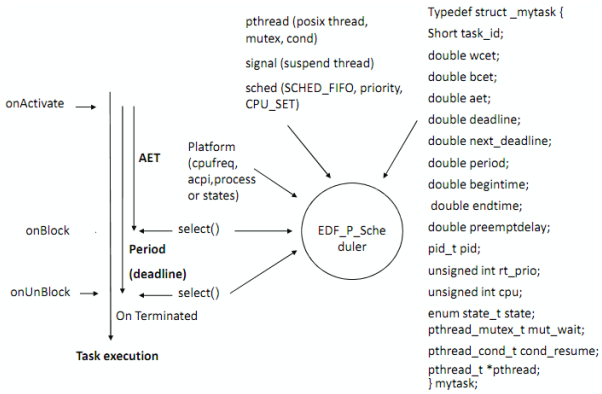

Figure 3: Scheduler architecture

5.1 Results

By adding monitoring data to the code, we have been able to rectify incorrect executions (e.g. improper shutdown of a task, exceeding the period, etc.). As a result, we succeeded in establishing a scheduling policy executing from the Linux user space that is dynamic, preemptive and accurate. The time overhead added by the scheduler never exceeds 0.01second. Indeed, a high solicitation could affect the real-time capabilities of the system that may not respond within the time limit. Taking into account the 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. Moreover, the tasks have to meet the condition of schedulability of equation (2) to be valid.

As a matter of fact, an aspect of parallelism appears during the execution and offers a high level of performance. The obtained results respect the priorities allotted to each task and are conform to the EDF policy: every task changes its priority proportionally with the approach of its deadline. Being multiprocessor, our strategy of scheduling will be given the responsibility to distribute these tasks between two processors. Besides, the scheduler attempts to schedule the same processes on the same processors for as long as possible in accordance with the processor affinity. We have assigned to the first processor the first two tasks and to the second one the other tasks. The result is that the task 1 and task 2 run only on CPU 1 and the other processes run on the other processor. Moreover, we compare the results of the execution with the parameters of the tasks already established. We raise the state of the task and we announce also the process ID (each process is represented by a unique identifier: PID), the priority, its start and the end time. This approach enabled us to have a comprehensive view of the scheduler. Equipped with a dynamic priority, if a process is running with a high priority the scheduler will

immediately preempt the running process, and switch to the newly runnable process. When a task ends and the field “onTerminated” is set to 1, the ready list changes involving a new sequence in accordance with the foundations of the EDF algorithm. Without changing any parameter of the scheduler, it should be noted that the results are consistent from one simulation to another. This shows the correctness of our implementation choices in particular for preemptions.

5.2 Simulation with STORM

To verify the scheduling results, we have used a simulation tool called STORM (Simulation TOol for Real time Multiprocessor scheduling)[23][24]. The original need to develop STORM came from the works of the IRRCyN research unit[1] .

This simulator considers the requirements of tasks, the characteristics and execution conditions of hardware components and the scheduling rules. Depending on the scheduling policy and the resources described in a XML file, it runs every task over a specified time interval [4]. The results of the simulation are a set of diagrams as illustrated in figure 4. All these diagrams permit analyzing the behavior of the system (tasks, processors, timing, performances ...).

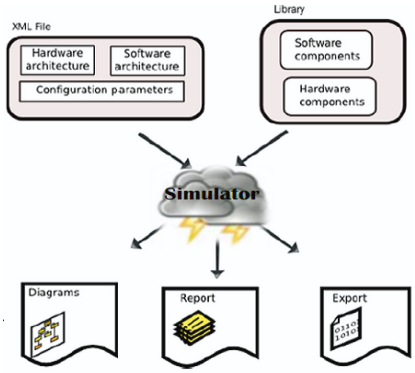

Figure 4: The STORM simulator [6]

A window displays a Gantt diagram of every task over an interval from 0 to 50 (default values). The title of the window refers to the name given to the task in the XML file (PTASKT1, PTASKT2…). In two other diagrams, we can observe the tasks assigned to processors CPUA and CPUB over the same interval. We can verify the allocation of tasks on processors according to their availability and to the priorities. 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 low priority processes to be preempted. Preemption helps also, to satisfy the constraints especially the real time constraint.

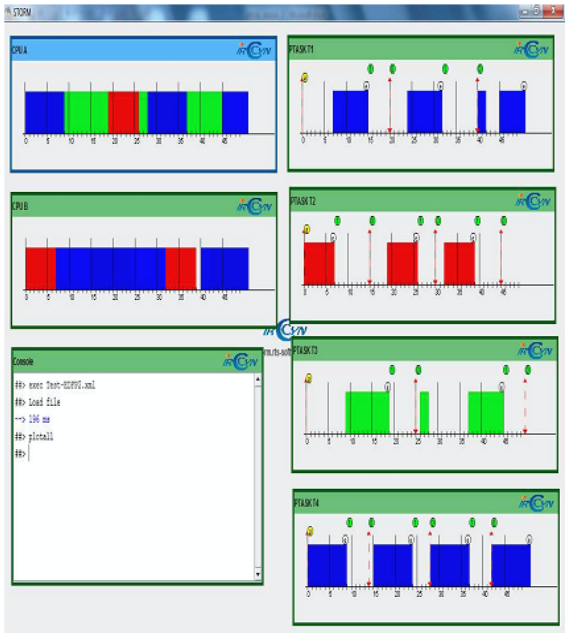

Figure 5: Results with STORM

Moreover, STORM allows multiprocessor simulation and analyzes energy consumption based on estimations. This simulator also provides support for DPM techniques but not DVFS. Being under development, a preliminary study of this tool has been necessary to determine its operation before any action of implementation. To facilitate the development and the checking of the best performance, the development of the scheduler was based on the specification of the EDF scheduler of STORM. Indeed, we tested its performances by considering four tasks as it is described in figure 5. Sorting the queue of the ready tasks will be in ascending order of deadlines which are characteristic of tasks entered in the XML specification. Indeed, the first activation (onActivate) and the following activations (onUnblock) will induce an addition of the corresponding task to the ready queue, whereas the events of termination of jobs (onBlock) or of task (onTerminate) correspond to a rejection of the corresponding task from the ready tasks queue. According to this first evaluation, we note that STORM checks initially the utilization ratio of processors and then assign the tasks so that they can run in parallel. In the simulation, we consider that all processors are identical and of type”CT11MPCore”. However, we note that some tasks switch from one CPU to another unlike their execution with Linux such as the task 3. The STORM EDF scheduler follows the EDF policy but it proceeds differently.

[1] The development of STORM came from the works of the PHERMA research project (”Parallel Heterogeneous Energy efficient Real-time Multiprocessor Architecture” (see http://pherma.irccyn.ec-nantes.fr).

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

The most substantial challenge of designers in wireless sensor networks remains always how to reduce the energy consumption in order to maximize the lifetime of the nodes. Therefore, we were interested in ensuring a judicious sharing of the energy resources through the DVFS and DPM techniques. We developed an EDF scheduler working in the "user space" level under Linux, so that we can extend it to the use of these techniques and to be able to try the strategies and to measure the energy gain on some boards. Besides, the time management and the 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 executing schedules where deadlines are met. Thus, scheduling algorithms provide a reliable mean to save the power consumption by trading off the energy against the fidelity and predicting the computation requirements of each task. 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 major advantage of Linux is the availability of many API that facilitates the development and make experimentation easier on boards. It guarantees also 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 correctness of execution with the simulation multiprocessor scheduling tool STORM. A futuristic approach would be to consider an algorithm which applies a couple of voltage and speed to the processor depending of the state of tasks in the system. Future works will be to extend this scheduler to exploit DVFS and DPM techniques. This will permit evaluating the power by measuring power consumption on concrete development board.

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 », National Symposium of GDR SOC-SIP, 2007.

[3] David Decotigny. « Bibliographie d'introduction à l'ordonnancement dans les systèmes 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] STORM: 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», PhD. Thesis, Institut National Polytechnique of Grenoble, Avril 2008.

[9] Andrea Castagnetti, Cécile Belleudy, Sébastien Bilavarn, Michel Auguin, "Power consumption modeling for DVFS exploitation", 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools,p579-586, 2010.

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

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

[12] 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.

[13] Amit Sinha, Anantha Chandrakasan "Dynamic Power Management in Wireless Sensor Networks", IEEE Design & Test of Computers, April 2001.

[14] Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, Ro Bogliolo, and Giovanni De Micheli. "Policy Optimization for Dynamic Power Management". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18 :813,833,1999.

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

[16] 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.

[17] 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

[18] 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.

[19] TinyOS: http://www.tinyos.net/

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

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

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

[23] Richard Urunuela, Anne-Marie Déplanche, Yvon Trinquet "simulation for multiprocessor real-time scheduling evaluation", EUROSIM, Prague,2010

[24] Richard Urunuela, Anne-Marie Déplanche, Yvon Trinquet " A Simulation Tool for Real-time Multiprocessor Scheduling Evaluation", Emerging technologies and factory Automatism (ETFA), Spain, 2010