Scheduling and Fault-Tolerance with Free Open-Source Components for Real-Time Applications

M. Silly-Chetto[1], A. Marchand[1], T. Garcia[1] and C. Plot[3]

[1] IRIN / University of Nantes Rue Christian Pauc 44306 Nantes cedex 03 France
maryline.silly@iut-nantes.univ-nantes.fr
[2] CRTTI / IUT of Nantes La Chantrerie CP3003 44087 Nantes cedex 03 France
plot@iut-nantes.univ-nantes.fr

In this paper, we will describe a national project (work supported by the French research office) that aims at the improvement of embedded computing systems for applications with real-time constraints. The objective of this project is first to create a library of free software components for the design of real-time operating systems and second, to participate in the evolution of an opened community standard, Linux. The key objective is as well to demonstrate the applicability and the interoperability of these software components by simulation, by integration and finally by tests on a real application. The tests on a mobile robotic platform (an Automated Guided Vehicle) are performed to show the benefits in terms of both improved integration process and adequacy with strict requirements on safety and reliability of next generation applications. First, strategies to equip Linux with real-time are presented and illustrated with the open-source real-time system LINUX/RTAI. Then, we are interested in the CLEOPATRE project itself and more precisely in its objectives in terms of providing dynamic scheduling and timing fault-tolerance. In addition, we tackle the problem of overload management with a brief description of the existing scheduling schemes before introducing the library of free software components. Finally, we present the CLEOPATRE demonstrator used to prove the efficiency of the new Linux-based operating system.

1 Real-Time under Linux

1.1 The Open-source model

The open-source sotftware concept was born, there is a score of years, in the academic research centers. Since its beginning, the research community has observed an exponential evolution of the number of users mostly thanks to the Internet diffusion. Indeed, the concept allows, both the user and the developer, to make copies of the program and distribute those copies, to have access to the source code and to make improvements to the program. In such a way, users profit from a free software that can be easily modified in order to fulfill specific needs. For the developers' point of view, they provide code pieces, test and make bug corrections, which is a manner of being recognized by their pars.

1.2 Providing real-time functionalities to Linux

Three strategies need to be combined in order to equip Linux with real-time. The first one consists in making the Linux kernel fully preemptable. The second one is provided by adding a real-time scheduling to the existing one. The last one carries out interrupt management changes. Real-time Linux extensions are a good solution while they integrate new real-time functionalities, thus representing a real extensibility for the original Linux kernel. Two major projects can thus be quoted : RTLinux representing the original real-time Linux API and RTAI which is a derivative of RTLinux.

1.3 The RTAI solution

An illustration of such an open-source real-time operating system is RTAI (Real-Time Application Interface), which has been chosen by the CLEOPATRE team for its simple adaptation and lower intrusion in the Linux kernel. Indeed, RTAI lies on a clear and efficient abstraction model : the RTHAL concept as illustrated below.

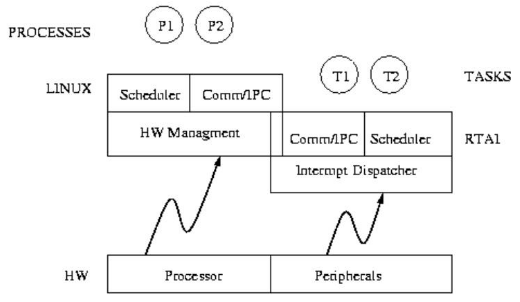

Fig. 1. The Real-Time Hardware Abstraction Layer [LM00]

As a matter of fact, the additional layer basically consists in an interrupt dispatcher and it is important to note that, in any real-time application built under RTAI, Linux is considered as the idle task.

2 The CLEOPATRE project

2.1 Objectives of the project

The main objective of Cleopatre project (Open Components for Embedded Real-time Applications) is to create a library of free-software components in order to develop real-time systems. To facilitate development of a real-time software, the user needs a framework that includes kernel components and tools that help him in choosing the good components for his application.

Cleopatre will produce a kernel that will be composed by a Linux kernel together with a modified RTAI-Linux extension.

Normally, every real-time application is composed of multiple tasks with different levels of criticality. A task can be hard real-time when its deadline must be met absolutely or soft real-time when losing its deadline is tolerated. Within the Cleopatre project, we consider that every real-time application software can include these two kinds of tasks which will be run at the same time with the same kernel.

Both RT-Linux and RTAI have become mature real-time operating systems which are ready for serious industrial use. However, the only scheduling policy implemented in RT-Linux and RTAI-Linux is the Rate Monotonic Algorithm (RMA) in which every task is assigned a fixed priority according to its period. Such scheduling policy can be efficient in safety critical systems where no event are unpredicted, all the tasks are periodic and are hard real-time. Dynamic approaches have many advantages over RMA because they are particularly appropriate to systems that include tasks with different levels of criticality.

In Cleopatre, we are concerned about providing one framework which permits to develop hard, soft or both real-time applications using a library of selectable components dedicated to dynamic scheduling, resource control access, fault-tolerance. So, new services not available in current real-time operating systems are delivered as a structured set of components, each one beeing responsible for providing a new functionality. This is performed by means of modifying current RTAI or adding new software layers over them using the current API. As starting point, we use a standard Linux kernel version 2.2.14 and the RTAI-Linux extension version 1.3.

2.2 Dynamic scheduling services

Preemptive scheduling of periodic tasks

A well-known fixed priority-scheduling algorithm is the deadline-monotonic algorithm (DM), which executes first the task with the shortest relative deadline. DM is optimal among fixed priority schedulers. The Earliest Deadline algorithm (EDF) executes tasks in the order of their absolute deadline EDF is optimal among all scheduling algorithms. While by the criterion of schedulable utilization, EDF outperforms DM. However, the advantage of DM is the predictability of which tasks will miss their deadline during an overload.

In a hard real-time system, it is necessary to perform an off-line test (a schedulability test) for the purpose of validating that the application can meet all its hard deadlines according to a chosen algorithm. When the tasks are independent, sufficient schedulability tests exist both for EDF and DM. These two schedulers have been integrated in RTAI-Linux.

Aperiodic and sporadic task servers

An aperiodic task is a non-periodic task with no deadline, while a sporadic task is a non-periodic one with a hard deadline.

The objective of the scheduler is:

- To decide whether to accept or reject the newly occurred sporadic task and to schedule the task with all the others.
- To complete each aperiodic task as soon as possible without causing periodic tasks and accepted sporadic tasks to miss their deadlines.

Three alternative approaches have been considered for implementation. According to the background server (BGS), aperiodic and sporadic tasks are scheduled and executed only at times when there is no periodic task ready for execution. According to the Total Bandwidth Server (TBS) [BS96], the server has a fixed size budget and fixed period but the server deadline is dynamically adjusted so as to replenish its budget early . The Slack Stealing approach consists in dynamically computing slack in order to keep track of the amount of available processor time for executing the aperiodic tasks as soon as possible and guaranteeing the deadlines of sporadic tasks whenever possible. The EDL server (EDLS) is a dynamic slack stealer, which has been integrated in Linux. See [Liu00] for detailed description of these approaches.

Resource allocation policies

A resource allocation policy is a set of rules that govern when and under what conditions each request for resource is granted and how tasks requiring resources are scheduled. The simplest way to control access of resources is to schedule all critical tasks non-preemptively which amounts to execute the task holding a resource at the highest priority. We call this protocol the Super-Priority Protocol (SPP). According to Priority Inheritance Protocol (PIP), any task that holds a resource and provokes blocking for another task inherits the priority of the blocked task until it releases the resource. The Priority Ceiling Protocol (PCP) is an extension of PIP to prevent deadlocks and reduce the blocking time. It makes use of a parameter associated to every resource, called ceiling priority. See [SRL90] for more details. SPP and PIP have been integrated while PCP is being in test.

2.3 Timing fault-tolerance

Time redundancy is an effective method in order to maintain the properties of correctness and timeliness of a real-time system even in the presence of faults. Among the error recovery methods, we have considered the Deadline Mechanism where each fault-tolerant task is characterized by two different copies (primary and backup) [LC86]. The aim of the scheduler is to guarantee the execution of the backup process whenever the primary one fails. And the processor time reserved for the execution of the backup process is reclaimed if the primary process executes sucessfully. Such a fault-tolerant scheduler has been integrated in Linux/RTAI.

2.4 Overload management

Real-time systems are systems where correctness is determined by both temporal and functional correctness, i.e., in simple terms, a real-time system should deliver correct results without missing important deadlines. To achieve this goal, system architects typically attempt to anticipate every eventuality and design the system to handle all of these situations. In reality, however, unanticipated emergency conditions may occur, the processing required to handle the emergency may exceed the system capacity, thereby resulting in missed deadlines. The system is said to be in overload and the correct behaviour of the whole may be jeopardized. Consequently, the approach here consists in understanding the behaviour of ressource allocation algorithms under overload conditions. That's why, we will describe three scheduling schemes contributing to handle situations where the system becomes temporarily overloaded.

Best-effort scheduling

Best-effort schemes include those algorithms with no predictions for overload conditions. A new task is always accepted into the ready queue so the system performance can only be controlled through a proper priority assignment. In case of overload, the tasks with the minimum value density are removed. Best-effort scheduling is more appropriate to soft real-time environments.

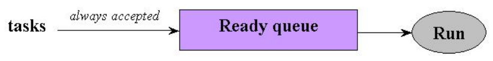

Fig. 2. Best-effort scheme

Guarantee scheduling

In guarantee algorithms, the load on the processor is controlled by an acceptance test executed at each task arrival. If the task set is found schedulable, the new task is accepted, otherwise, it is rejected. In case of overload, the main disadvantage lies in the fact that the new task is always rejected because task importance is neglected.

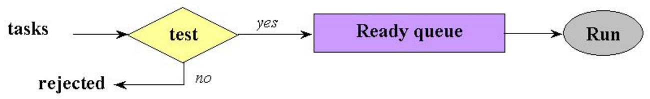

Fig. 3. Guarantee scheme

Robust scheduling

Timing constraints and importance are separated by considering two different policies : one for task acceptance and one for task rejection. These algorithms prove more effective in practical situations with tasks of various importances. We can note that, during a transient overload, a new important task that would have been rejected in the guarantee scheme, could be accepted in the robust scheme.

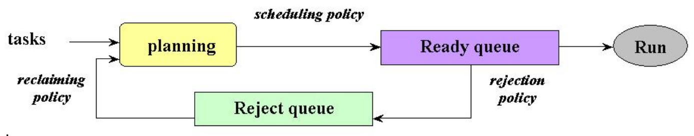

Fig. 4. Robust scheme

3 Description of the library

The native RTAI scheduler has been transformed into a background task of the new scheduler. RTAI maintains a list of blocked tasks descriptors. At each clock signal, the timer interrupt handler keeps up to date the system clock and looks for the waiting tasks that must be released to make them ready.

EDF and DM have been implemented. RTAI root has been maintained and adapted to build new schedulers.

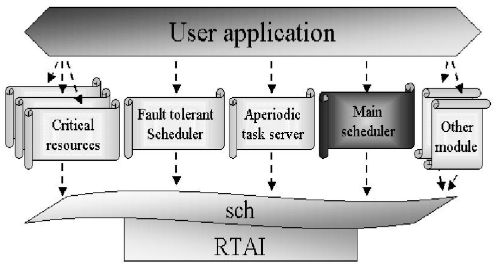

Fig. 5. Description of the library [SGL03]

4 The CLEOPATRE demonstrator

The Cleopatre demonstrator consists in a bidirectional mobile robotic platform fitted with several embedded navigation sensors. Its function is to ensure the routing of standard industrial plastic vats within a FMS (Flexible Manufacturing System), which represents a typical real-time concurrent system. Consequently, the robot control involves deadline-driven activities, resources allocation and real-time decision making.

A first phase in our experimentation was to control the mobile robotic platform using the native LINUX/RTAI operating system with fixed priorities [MPS02]. The second test phase consists in measuring performance gain (in terms of deadline guarantee) obtained by integration of the new open-source components librairies.

5 Conclusion

To date, progress is as follows : a version of the deadline mechanism has been implemented providing timing fault-tolerance for soft real-time systems. An other implementation based on optimal time recovery will be considered for hard real-time systems. The priority ceiling protocol is currently under integration in order to provide a synchronization mechanism preventing both deadlock and transitive bloking.

A measure of all the overheads induced by the interrupt latency, by the context switches and by the different mechanisms which are implemented in the library, will be performed. The efficiency evaluation of this new Linux-based operating system will be deduced by a series of tests in direct connection with the Cleopatre demonstrator.

References

[BS96] Spuri, M., Buttazzo, G.: Scheduling Aperiodic Tasks in Dynamic Priority Systems, Journal of Real-Time Systems, Vol. 10, No. 2 (1996)

[LC86] Liestman, A.L., Campbell, R.H.: A fault-tolerant scheduling problem. IEEE Trans on Soft. Eng., Vol. 15, No. 10, pp. 1089-1095 (1986)

[Liu00] Liu, J.W.S.: Real-Time Systems, Prentice-Hall (2000)

[LM00] Lineo Inc., Mantegazza, P.: RTAI Programming Guide 1.0. Lindon, Utah (2000)

[MPS02] Marchand, A., Plot, C., Silly, M.: Real-time mobile robot navigation with LINUX/RTAI. In Proc. IEEE International Conference on Control and Automation. Xiamen, Chine (2002)

[SGL03] Silly-Chetto, M., Garcia, T., Lucas, G., Orhant, Y.: On using Real-time Linux in a mobile robot application. 7th World Multiconference on Systemics, Cybernetics and Informatics. Orlando, USA (2003)

[SRL90] Sha, L., Raykumar; R., Lehoczky, J.P.: Priority inheritance protocols : An approach to real-time synchronisation. IEEE Trans. on Computers, Vol. 39 (1990)