Computers and Electrical Engineering 37 (2011) 365–375

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Periodic timers revisited: The real-time embedded system perspective ☆

Antônio Augusto Fröhlich *, Giovani Gracioli, João Felipe Santos

Software/Hardware Integration Lab, Federal University of Santa Catarina, 88040-900 Florianópolis, SC, Brazil

ARTICLE INFO

Article history:
Received 24 March 2010
Received in revised form 21 February 2011
Accepted 2 March 2011
Available online 26 March 2011

ABSTRACT

Common sense dictates that single-shot timer mechanisms are more suitable for real-time applications than periodic ones, specially in what concerns precision and jitter. Nevertheless, *real-time embedded systems* are inherently periodic, with tasks whose periods are almost always known at design-time. Therefore a carefully designed periodic timer should be able to incorporate much of the advantages of single-shot timers and yet avoid hardware timers reprogramming, an expensive operation for the limited-resource platforms of typical embedded systems.

In this paper, we describe and evaluate two timing mechanisms for embedded systems, one periodic and another single-shot, aiming at comparing them and identifying their strengths and weaknesses. Our experiments have shown that a properly designed periodic timer can usually match, and in some cases even outperform, the single-shot counterpart in terms of precision and interference, thus reestablishing periodic timers as a dependable alternative for real-time embedded systems.

1. Introduction

The notion of time is essential to any real-time embedded system. The system needs to keep track of time flow to schedule tasks and also to provide time services, such as delays and alarms, to applications. Historically, operating systems (OS) have been implementing time management based on a hardware timer that is configured to periodically trigger interrupts, thus giving rise to a system time unit called *tick*. Ticks define the minimum perception of time flow within the system and rule every sort of time-driven events. The mechanism is usually implemented with a single hardware timer, whose interrupt handler is overloaded with operations around task scheduling and timed event propagation.

Though well-accepted in the realm of general-purpose systems, time management strategies based on periodic timers face strong criticism from the real-time system community [1]. For instance, in a system with a periodic timer configured to generate 10 ms ticks, a 15 ms delay request may result, in the worst case, in a waiting time of 30 ms. Fig. 1 exemplifies this worst-case scenario. The request is posted just after a tick is generated, thus counting will only start on the next tick. Moreover, 15 might be rounded up to 20 (multiple of a tick), yielding a total time of 30 ms. In addition to the lack of precision in time services, the periodic timer handler is constantly activated, even if no action needs to be taken, causing overhead and interference on running tasks. These limitations fostered the mechanism of *single-shot* timer, with dedicated timers being programmed to fire exactly when an action has to be performed [1–3].

Nonetheless, despite these unquestionable issues about periodic timers, while performing experiments in the context of a previous paper [4], we realized that single-shot supremacy might be indeed more closely connected to implementation issues than to the concept itself. Some of the aspects that called our attention were:

☆ This work was partially supported by the Coordination for Improvement of Higher Level Personnel (CAPES) grant, projects RH-TVD 006/2008 and 240/2008.

* Corresponding author. Tel./fax: +55 48 3721 9516.
E-mail addresses: guto@lisha.ufsc.br (A.A. Fröhlich), giovani@lisha.ufsc.br (G. Gracioli), jfsantos@lisha.ufsc.br (J.F. Santos).

0045-7906/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2011.03.003

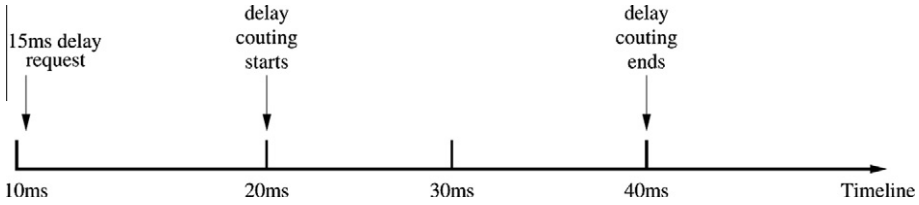

Fig. 1. An example of a worst case scenario for a delay request using a periodic timer.

- Real-time embedded systems are intrinsically periodic. Even simple software architectures, such as cyclic executives, have a period derived from the main loop length. Real-time scheduling in more complex embedded systems is essentially periodic (e.g. Earliest Deadline First, Rate Monotonic, etc).
- Single-shot timers are usually multiplexed on a few physical timers, demanding constant hardware reprogramming, what, in some systems is much slower than software reprogramming.
- Single-shot timers, in practice, do overflow, demanding some sort of periodic fall-back mechanism.

In this way, a carefully designed periodic timer mechanism could, in theory, incorporate much of the advantages of single-shot timers. Indeed, even non real-time OS now feature improved periodic timers: WINDOWS VISTA can adjust the period of the hardware timer according to the system load [5]; LINUX delivers a secondary high resolution timing interface to applications [6]. Therefore, in this paper we investigated this hypothesis by comparing the behavior of both time management strategies (i.e., periodic and single-shot) in the same scenario (i.e., hardware platform, OS, and applications). Our results confirmed that a properly configured and implemented periodic timer may yield high precision and low overhead. Single-shot timers, on the other hand, are usually able to match the precision of periodic timers, and can cause less interference in the system when a periodic timer is not well-configured. The design and implementation considerations presented in this paper can be incorporated by virtually any OS. They can also serve as guidelines for other groups developing single-shot timer mechanisms on the premise that this would eliminate all the trouble caused by periodic timers on real-time applications.

The remainder of this paper is organized as follows: Section 2 presents related work; Section 3 presents the design and implementation of single-shot and periodic timing mechanisms; Section 4 describes the experimental evaluation of proposed mechanisms, along with a comparative analysis; Section 5 closes the paper with our final considerations.

2. Related work

Tsafrir et al. approach some important issues around periodic time management, particularly lack of precision and power consumption, as they introduced the concept of *Smart Timer* [1]. The proposed timer mechanism can be summarized by three main properties: (1) accurate timing with configurable maximum latency; (2) reduced management overhead by triggering combined nearby events, and (3) reduced overhead by avoiding unnecessary periodic events.

Kohout presents a strategy to efficiently support RTOS using core components implemented in hardware [7]. The main objective is to reduce the impact caused by the RTOS on applications, which is assessed in terms of response time and CPU usage. The work introduces the Real-Time Task Manager (RTM), a hardware component that deals with task scheduling, time management, and event management. The RTM support for time management functions causes a 10% reduction on CPU time (with 24 tasks).

Aron and Druschel introduced the concept of *Soft-Timer*, which triggers the time manager on the return of every system call [2]. Hardware timer interrupts are simultaneously deployed to ensure minimum timing requirements for applications. Their measurements show a reduction in the number of context switches and an increase in the time service precision. This is mainly attributed to the fact that system calls tend to be more frequent than timer interrupts. Nonetheless, periodic timer interrupts are still necessary, since the rate of system calls is unpredictable.

Firm Timer combines three different time management approaches, single-shot timer, soft-timer, and periodic timer, to build an efficient, high-resolution, low-overhead mechanism [3]. The mechanism uses an "overshooting" parameter S, that is added in a requested time T (T + S). If the kernel is called by a system call, interrupt, or an exception after T, but before $T + S$, then the requested period is handled (*Soft-Timer* approach), and the overhead associated to the interrupt handler at $T + S$ is avoided. The authors claim that in some cases the kernel is called after a timer has expired but before the timer generates an interrupt. The combination of the three time management approaches allows a reduction in the number of timer interrupts, thus reducing the overall overhead of periodic timers in the system. However, as in the *Soft-Timer*, there is an extra overhead associated to polling and checking for timers at each soft-timer point [3]. Moreover, the overshooting parameter causes a lack of timing precision, which compromises its use in hard-real time applications.

L4-EMBEDDED microkernel implements time management using a traditional periodic timer scheme, but exempts the interrupt handler from duties such as time-out management and time-of-day keeping. Those activities are performed by specific servers at user-level. This work distribution improves the response time of the interrupt handler [8]. Single-shot timer mechanisms are to be used in future versions of the system aiming at energy consumption optimization.