Vol.2010-EMB-18 No.2
2010/8/9

Integrated Scheduling in a Real-Time Embedded Hypervisor

Daniel Sangorrin,[†1] Shinya Honda[†1]
and Hiroaki Takada[†1]

Real-Time hypervisors are useful to support the concurrent execution of a general-purpose operating system (GPOS) and a real-time operating system (RTOS) in isolation. However, the integrated nature of real-time embedded systems requires a global approach to scheduling. This paper presents the design and implementation of an integrated scheduling architecture for a real-time hypervisor. The proposed approach makes it possible to mix the priority of each activity in the system (i.e., RTOS and GPOS tasks and interrupt handlers) from a global point of view. Furthermore, the implementation provides the mechanisms to protect the RTOS activities from any potential misbehavior of the activities in the GPOS.

1. Introduction

Several embedded markets are currently facing the challenge of consolidating real-time applications and high-level information software (e.g., a web browser or media player) into a single platform to reduce product costs[1)–3)]. In order to develop high-level information software efficiently, the use of a general-purpose operating system (GPOS) and its libraries is generally essential. On the other hand, most GPOS cannot satisfy the strict requirements imposed by real-time applications in terms of security, reliability and determinism[2)]. For instance, security holes are discovered continuously in GPOS such as Windows® or GNU/Linux[4)]. For that reason, in 5) Nakajima et al. presented SafeG (Safety Gate), a reliable hypervisor based on common embedded security hardware (ARM TrustZone®[6)]), which supports the concurrent execution of a real-time operating system (RTOS) and a GPOS.

This paper extends the functionality of the SafeG hypervisor through an integrated scheduling architecture. The proposed architecture supports priority interleaving for each activity in the system (i.e., RTOS and GPOS tasks and interrupt handlers) from a global point of view. It makes it possible to enhance the responsiveness of the GPOS activities while preserving the timeliness of the RTOS activities even in the presence of GPOS execution time overruns thanks to the use of resource reservations.

The paper is organized as follows. Section 2 reviews, briefly, the architecture and characteristics of SafeG, a dual embedded hypervisor based on the ARM Trustzone technology. Section 3 describes the architecture for scheduling the two guest operating systems in an integrated fashion. A prototype implementation is described in Section 4. Section 5 compares the presented study with previous work. Finally, Section 6 draws some conclusions and discusses future work.

2. SafeG hypervisor

The SafeG (Safety Gate) hypervisor was originally presented in 5) as a reliable virtualization architecture to execute concurrently an RTOS and a GPOS (in the current implementation TOPPERS/ASP and GNU/Linux are supported) on an embedded single processor. The architecture takes advantage of the ARM TrustZone technology[6),7)] which introduces the concept of *Trust* and *Non-Trust* states. Trust state provides similar behavior to existing privileged and user mode levels in ARM processors. On the other hand, code running under Non-Trust state, even in privileged mode, cannot access memory space (devices included) that was allocated for Trust state usage, nor can it execute certain instructions that are considered critical.

In order to control the TrustZone state, a new mode, called Secure Monitor mode, has been added to the processor. Switching between Trust and Non-Trust state is performed under Security Monitor mode by SafeG with interrupts disabled. The overall organization of the system is depicted in **Figure 1**. Spatial isolation of the RTOS is supported by configuring resources (memory and devices) used by the RTOS to be accessible only from Trust state. The remaining resources are configured to be accessible both from Trust and Non-Trust state. Time isolation of the RTOS activities is supported by carefully allocating the two types of interrupt. FIQ interrupts are forwarded to the RTOS, while IRQ interrupts are forwarded to the GPOS. In Trust state, GPOS interrupts are disabled so that the

†1 Graduate School of Information Science, Nagoya University

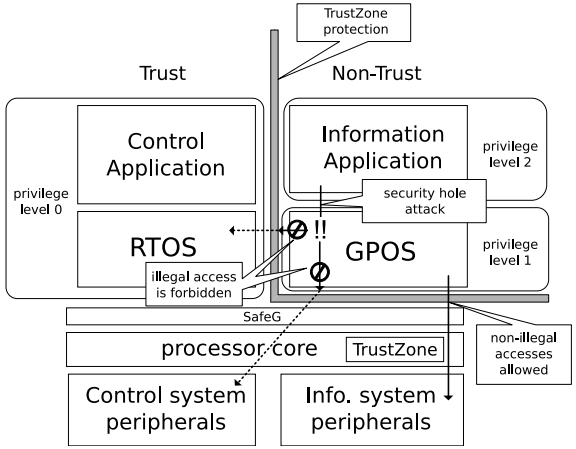

Fig. 1 SafeG: hypervisor based on ARM TrustZone

GPOS cannot affect the execution of the RTOS. For that reason, the GPOS can only execute once the RTOS makes an explicit request, through a Secure Monitor Call (SMC), to SafeG. On the other hand, during the GPOS execution, RTOS interrupts are enabled so that the RTOS can recover the control of the processor (e.g., through the interrupt associated to the RTOS system timer). TrustZone is configured to prevent the GPOS side from disabling RTOS interrupts.

2.1 Idle scheduling problem

A virtualization architecture designed for real-time embedded systems must provide a deterministic scheduling algorithm. Most real-time hypervisors schedule the two operating systems as black boxes in a way that the GPOS is only executed once the RTOS becomes idle[8),9)]. This method allows the RTOS to take precedence over the GPOS and thus preserve its determinism. However not all RTOS activities require the same degree of responsiveness[10)], and some GPOS applications and interrupt handlers, such as multimedia on mobile wireless devices, require a certain quality of service[11)].

Figure 2 highlights the disadvantages of scheduling the GPOS as the idle thread of the RTOS. In the figure, an interrupt request to the GPOS gets delayed until all the processing at the RTOS is finished. In the worst case, the interrupt

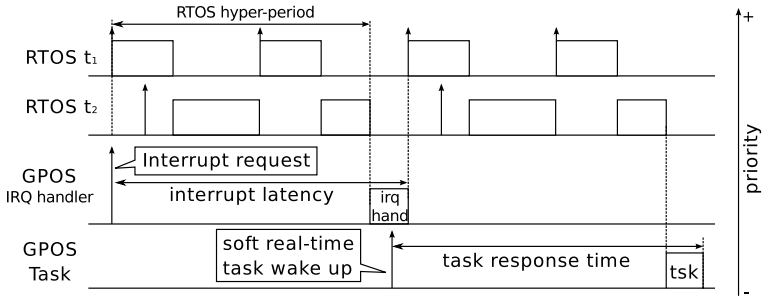

Fig. 2 GPOS as idle task

request will be attended only after a hyper-period in the RTOS schedule. The latency that can be achieved with this method may not be enough for certain devices. The same situation can happen for the GPOS soft real-time tasks which require a certain quality of service to work correctly.

In the first version of SafeG[5)], the GPOS was only executed once the RTOS would release the processor. More in detail, the RTOS idle task was programmed to call SafeG (using an SMC) in order to perform a switch to the GPOS. In order to tackle the disadvantages of the idle approach, an integrated scheduling approach is presented in the following section.

3. Integrated scheduling architecture

This section describes the integrated scheduling architecture aimed at providing the mechanisms to configure the priority at which each activity in the system (i.e., interrupts and tasks) is executed on a global basis.

Figure 3 shows the overall organization of the system. Each group of GPOS activities is represented by a certain LTASK (Latency Task) task in the RTOS. An LTASK task can represent a single activity or a group of them and the number of LTASK tasks is only limited by the total number of activities in the GPOS. LTASK tasks are all executed under an aperiodic server policy with a configurable priority, budget and period, except for the BTASK (Background Task) task which is equivalent to the RTOS idle task. The body of each LTASK task consists of a loop executing a world-switch request to SafeG. It is worth mentioning that the presented architecture does not intend to modify the priority order at which

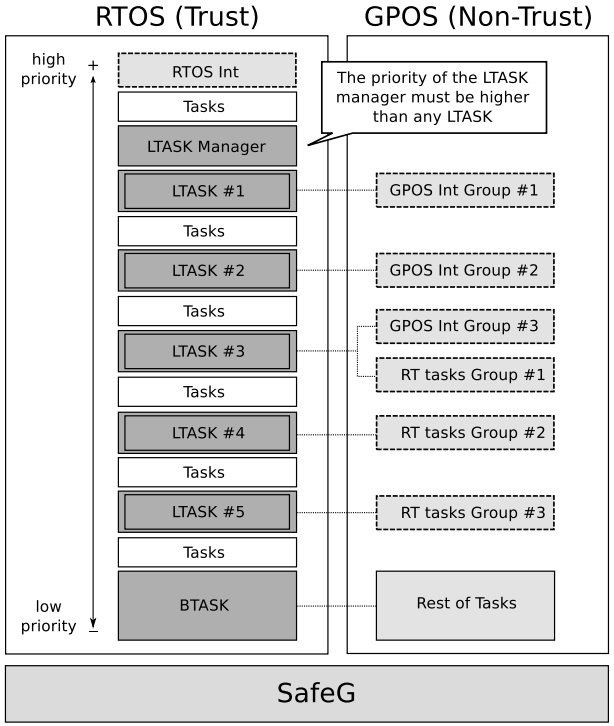

Fig. 3 Latency Tasks for fine granularity

tasks and interrupts are executed inside the GPOS. Therefore, activities assigned to a certain LTASK task must always execute at a higher priority range than activities assigned to a lower priority LTASK task. In particular, if several GPOS interrupts are assigned to different LTASK tasks they must be configured with different hardware priorities as well. For the same reason, whenever the budget of a certain LTASK task expires, the budget of the next LTASK task in decreasing priority order will be inherited. In other words, the neat mapping between GPOS activities and LTASK tasks depicted in Figure 3 only holds provided that the GPOS activities do not overrun the budget assigned to them. Otherwise they will steal budget that was originally assigned to an LTASK task representing a group of activities at a lower priority range. Should the execution of groups

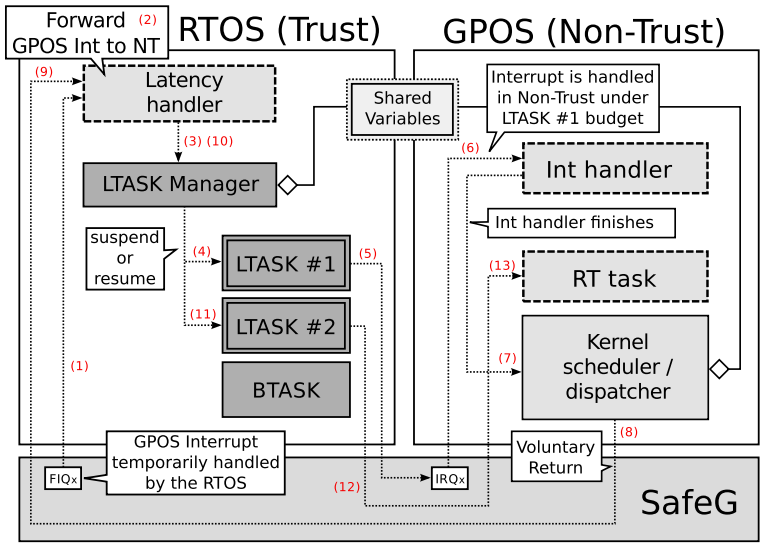

Fig. 4 LTASK manager activation

of activities be isolated from each other, the GPOS kernel must provide the necessary mechanisms (e.g., resource reservations) not the hypervisor.

3.1 Activation of the LTASK tasks

The activation and deactivation of the LTASK tasks is controlled by the so-called *LTASK manager*. The LTASK manager is a task in the RTOS that is configured with priority higher than any of the LTASK tasks. Each time the LTASK manager is awakened, it determines the GPOS activity that currently has the highest priority, resumes the LTASK task corresponding to that activity and suspends any LTASK task with higher priority. If the budget of the current LTASK task expires, the budget of the next LTASK task in decreasing priority order is consumed. If all budgets are expired, the background task will be executed instead.

3.2 Shared variables

The architecture makes use of two variables that are shared between the two operating systems as depicted in **Figure 4**.

- The `LTASK_#ID` shared variable is written by the Latency manager and read

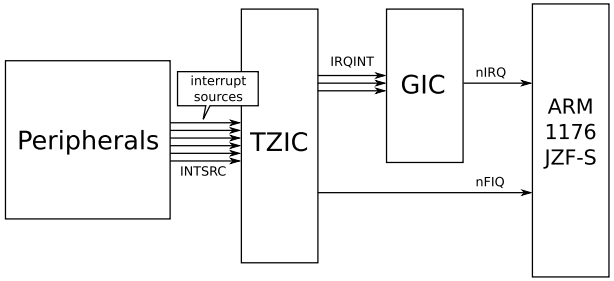

Fig. 5 TrustZone Interrupt Controller configuration manager activation

by the GPOS kernel. It indicates the identifier of the LTASK task that currently has the highest priority. It is used by the GPOS kernel at every scheduling decision in order to determine whether a voluntary return to the RTOS should be performed or not. For instance, if the kernel schedules a task that is assigned to an LTASK task with a different identifier than the one indicated by `LTASK_#ID` then, it returns the control back to the RTOS through SafeG.

- The `NT_Prio` shared variable is written by the GPOS kernel and read by the Latency manager. It indicates the priority of the task that is currently running in the GPOS. The LTASK manager reads this variable and the raw status of the GPOS interrupts in order to determine which LTASK tasks should be suspended and resumed.

3.3 GPOS interrupts flow

When a GPOS interrupt occurs, the Latency manager must be awakened to activate the corresponding LTASK task. For that reason, GPOS interrupts are temporarily handled in the RTOS before being forwarded to the GPOS. A method to achieve that is to leverage the features provided by the TrustZone Interrupt Controller (TZIC[12]). **Figure 5** shows a typical configuration of the TZIC, where interrupt sources (INTSRC) can be configured to produce an FIQ request to the RTOS or forwarded to the Generic Interrupt Controller (GIC) which produces an IRQ request to the GPOS.

Steps (1) to (6) in Fig. 4 and **Figure 6** show the execution flow when the Latency manager is awakened because the RTOS is notified about the occurrence of a GPOS interrupt. The GPOS interrupt is temporarily handled in the RTOS

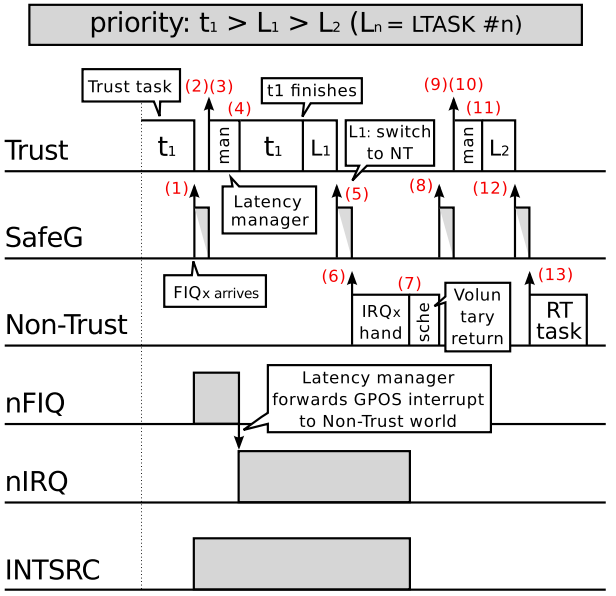

Fig. 6 LTASK manager activation timeline

by the so-called *Latency handler*. The Latency handler forwards all interrupts assigned to that interrupt's group to the GPOS (Non-Trust world) as illustrated in steps (1) to (2). Then, the Latency handler awakes the LTASK manager which in turn resumes the LTASK task (e.g. LTASK #1) assigned to that interrupt (steps (3) to (4)). When that LTASK task becomes the running task, it calls SafeG in order to switch to the GPOS where the interrupt is served (steps (5) to (6)).

3.4 GPOS interrupt voluntary return

Once the interrupt handler is finished, the GPOS may need to return the control to the RTOS in order to conserve the remaining LTASK budget. Steps (7) to (13) in Fig. 4 and Fig. 6 show the execution flow when the Latency manager is awakened because the GPOS voluntarily returns the control back to the RTOS. After the GPOS handler finished its execution in step (7), the GPOS kernel schedules and dispatches a task. If that task belongs to a different group of

activities, the kernel sends a notification to the RTOS through SafeG in the form of an artificial interrupt that is handled by the Latency handler (steps (8) to (9)). Similar to steps (3) to (7), the Latency handler awakes the Latency manager which in turn resumes the LTASK task assigned to the task (e.g., LTASK #2) in steps (10) to (13).

3.5 GPOS tasks voluntary return

The running task may change for a number of reasons: a new task with higher priority arrives; a task is awaken; or the running task gets blocked, etc. When a context switch is performed the architecture tracks the currently executing task in the `NT_Prio` shared variable and determines whether a voluntary return is needed or not. If the new task belongs to a group whose identifier is different from the current `LTASK_#ID`, a voluntary return to the RTOS needs to be performed to activate the corresponding LTASK task.

4. Implementation

This section explains details about an prototype implementation of the presented integrated scheduling architecture. Details about SafeG's implementation can be found in 5), 13). The architecture has been implemented on a PB1176JZF-S board, equipped with a TrustZone-enabled ARM1176jzf processor at 210Mhz. TOPPERS/ASP version 1.3.1 was used as the RTOS and the ARM Linux 2.6.33 kernel for the GPOS.

4.1 SafeG modifications

SafeG was extended with a new Secure Monitor Call (SMC). The new call makes it possible to generate an exception in the other operating system as a way to notify the occurrence of a certain event. The new call increased the size of SafeG's code by only 16 assembler lines. The same idea can be used to create a more generic communication mechanism between the two operating systems and is left for future research.

4.2 Linux kernel modifications

In order to support the voluntary return functionality described in Section 3, the Linux kernel has been extended with a new module. The implemented module includes some initialization code (e.g., shared variables initialization) and contains the table to map task and interrupt priorities to LTASK group identifiers. It also provides the so-called `switch_hook` function which decides whether a voluntary return is required or not and updates the shared variable `NT_Prio`. This hook has been inserted in the Linux scheduler (`sched.c`) and the Linux interrupt code (for ARM architecture) and it can be activated or deactivated from user space (with root privileges) through a boolean value in the `debugfs` filesystem. When the hook is deactivated, the behavior of the whole system resembles the one of the system presented in 13). The total number of source code lines (in C language) included in the current version of the voluntary return module is 121. The module is quite independent from the Linux version. The main maintenance difficulty may come from inserting the switch hook in the appropriate places in the Linux kernel. Fortunately, the `ftrace` kernel tracer already provides an interface to capture any scheduling event.

4.3 TOPPERS/ASP modifications

The TOPPERS project[14] follows the ITRON standard for real-time operating systems to produce high quality open-source software for embedded systems. ASP (Advanced Standard Profile kernel) is one of TOPPERS real-time kernels and is based on the μITRON4.0[15] specification with several extensions. The implementation of the integrated scheduling framework for the Trust side has been accomplished in user space by using the μITRON4.0 interface provided by ASP. The ASP kernel was only modified to provide μITRON4.0 overrun handlers, which were used to build deferrable servers for the LTASK tasks.

4.4 Configuration and deployment tool

Since the number of LTASK tasks and their parameters (i.e., priority, budget, period and group) will depend on the application, a configuration and deployment tool has been created. **Figure 7** shows the deployment flow of the system. First, the application code for TOPPERS/ASP and Linux (user space or kernel space) is designed, implemented and tested independently. Then, a graphical configuration tool is used to specify the parameters of the LTASK tasks that are suitable to schedule the whole system on a global basis. Task priorities are expressed by positive values. For interrupts, priorities are expressed by negative values. The configuration tool automatically generates the integrated scheduling code for the two operating systems using a template and the configured values. The code is then compiled and linked to ASP and the Linux kernel producing the final

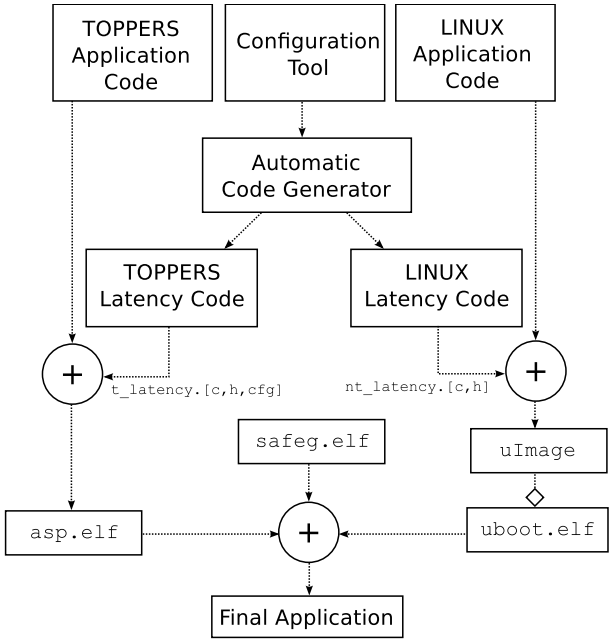

Fig. 7 Development flow

application.

5. Related work

In 11), a task grain scheduling algorithm for a virtualized embedded system was presented. The proposed architecture used the L4-embedded microkernel as a hypervisor running para-virtualized versions of Linux (Wombat) and TOPPERS/JSP (L4/TOPPERS). In order to implement task grain scheduling, each of the guest operating systems notifies the priority of the running task to the global scheduler (the L4 scheduler). Unfortunately, in that work the RTOS tasks are not protected from possible misbehaviors of the GPOS tasks. For example, it is not possible to detect an execution time overrun of a misbehaving GPOS task with high priority, and therefore RTOS tasks with lower priority may miss their deadlines, thus decreasing the reliability of the whole system. Compared to that work, the approach presented in this paper provides a method to protect the RTOS activities from the GPOS through the use of aperiodic servers. Furthermore, in addition to task-grained scheduling, integrated scheduling of the GPOS interrupts is considered. Finally, the proposed approach does not require the hypervisor to provide an scheduler since the RTOS scheduler is used instead. This helps to improve the verifiability of the hypervisor and eliminates the overhead in the priority notifications from the RTOS side.

6. Conclusions and future work

This paper introduced the design and implementation of a method to integrate RTOS and GPOS activities on top of a real-time hypervisor based on common embedded security hardware (ARM TrustZone®). The proposed approach provides a way to configure the execution parameters of each activity in the system from a global point of view. It makes it possible to provide a configurable quality of service to the GPOS soft real-time activities while the reliability and timeliness of the RTOS activities is preserved. A prototype implementation of the architecture on the ARM1176jzf processor was also presented. The changes required on the Linux kernel code were rather small.

As future work, an evaluation of the overhead and effectiveness of the proposed approach is planned. An intercommunications system and a porting to new multicore TrustZone processors will be also explored in the near future.

References

1) Heiser, G.: The Role of Virtualization in Embedded Systems, *1st Workshop on Isolation and Integration in Embedded Systems*, Glasgow, UK, ACM SIGOPS, pp. 11–16 (2008).
2) Heiser, G.: Hypervisors for consumer electronics, *CCNC'09: Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, Piscataway, NJ, USA, IEEE Press, pp.614–618 (2009).
3) Hergenhan, A. and Heiser, G.: Operating Systems Technology for Converged ECUs, *6th Embedded Security in Cars Conference (ESCAR)*, Hamburg, Germany, ISITS (2008).
4) Kinebuchi, Y., Koshimae, H., Oikawa, S. and Nakajima, T.: Virtualization techniques for embedded systems, *Proceedings of the Work-in-Progress Session: the 12th IEEE International Conference on Embedded and Real-Time Computing Systems*

and Applications, Sydney, Australia (2006).
5) Nakajima, K., Honda, S., Teshima, S. and Takada, H.: Enhancing Reliability in Hybrid OS System with Security Hardware, *The IEICE Transactions on Information Systems*, Vol.93, No.2, pp.75–85 (2010-02-01).
6) ARM Ltd.: *ARM Security Technology. Building a Secure System using TrustZone Technology, PRD29-GENC-009492C* (2009).
7) ARM Ltd.: *ARM1176JZF-S. Technical Reference Manual, DDI 0301G* (2008).
8) Cereia, M. Bertolotti, I.: Asymmetric virtualisation for real-time systems, *ISIE 2008*, Cambridge, pp.1680 – 1685 (2008).
9) Yoo, S., Liu, Y., Hong, C.-H., Yoo, C. and Zhang, Y.: MobiVMM: a virtual machine monitor for mobile phones, *MobiVirt '08: Proceedings of the First Workshop on Virtualization in Mobile Computing*, New York, NY, USA, ACM, pp.1–5 (2008).
10) Takada, H., Iiyama, S., Kindaichi, T. and Hachiya, S.: Linux on ITRON: A Hybrid Operating System Architecture for Embedded Systems, *SAINT-W '02: Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, Washington, DC, USA, IEEE Computer Society, pp.4–7 (2002).
11) Kinebuchi, Y., Sugaya, M., Oikawa, S. and Nakajima, T.: Task Grain Scheduling for Hypervisor-Based Embedded System, *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, Washington, DC, USA, IEEE Computer Society, pp.190–197 (2008).
12) ARM Ltd.: *AMBA3 TrustZone Interrupt Controller (SP890) Technical Overview, DTO 0013B* (2008).
13) Sangorrin, D., Honda, S. and Takada, H.: Dual Operating System Architecture for Real-Time Embedded Systems, *Proceedings of the Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2010)*, Brussels, Belgium (2010).
14) TOPPERS: Toyohashi OPen Platform for Embedded Real-Time Systems. `http://www.toppers.jp`.
15) Takada, H. and Sakamura, K.: "μITRON for small-scale embedded systems", *IEEE Micro*, vol. 15, pp. 46-54, Dec. 1995.