Hierarchical Scheduling for Integrating Real-time Applications with Interrupt Routines

Yutaka MATSUBARA, Shinya HONDA, *Nonmembers, IEEE,* and Hiroaki TAKADA, *Member, IEEE*

***Abstract*—This paper presents a split interrupt routine model for a two-level hierarchical scheduling in order to integrate multiple independently developed applications that consist of tasks and interrupt routines into a shared CPU. In this model, an interrupt routine is split into an Interrupt Handler (IH), providing device-depended service, and an Interrupt Service Task (IST), providing application-specific service. By using this model, a main part of an interrupt routine is handled as a task and become controllable by global EDF scheduling. We implement this model on an actual processor to evaluate the response time and overhead of activating an IST. Furthermore, we analyze the schedulability of the applications including delay caused by interrupt disabled time and propose a simple schedulability test method. This method helps system integrators, such as automotive manufacturers, to quickly determine which applications could be integrated to the system without detail knowledge of each application in the system design phase.**

I. Introduction

In recent years, system integrators of distributed real-time systems, such as an automotive control system, are challenging to reduce the number of CPUs in the system. To achieve this aim, multiple independently developed real-time applications are required to be integrated on a shared CPU. Furthermore, it is expected that all applications are schedulable on a shared CPU, if each application is schedulable on a dedicated CPU. For this purpose, hierarchical scheduling frameworks in [2]–[4] that provides temporal isolation among applications are suitable. However, these assumed that an application consists of only tasks, and ignored the existence of interrupt routines. In practice, when an interrupt occurs, an interrupt routine is immediately executed prior to all tasks regardless of the global scheduling policy. As a result, the prior work, which ignored the interrupt routines, could miss its deadlines. Figure 1 shows an example where a deadline miss is caused by the interrupt routine in a global EDF scheduling.

In this paper, we introduce a split interrupt routine model to a two-level hierarchical scheduling based on Bandwidth Sharing Server (BSS) algorithm in [2]. In the model, an interrupt routine is split into an Interrupt Handler (IH), providing device-depended service, and an Interrupt Service Task (IST), providing application-specific service. Because the IST can be handled as a normal task in the model, the effect of interrupt routines becomes controllable in a global scheduling. We evaluate the response time and the overhead of activating an IST through a prototype implementation. Furthermore, we present schedulability analysis with given interrupt disabled time. This result helps developers of automotive control system to integrate several ECUs (Electric Control Units) to one ECU and quickly determine which applications could be integrated to the ECU without detail knowledge of each application in the early system design phase.

The remainder of this paper is organized as follows. Section II presents the system model and the two-level hierarchical scheduling framework. Section III presents the interrupt routine model and gives the result of implementation and the schedulability analysis. Section IV presents related works. Section V describes conclusions of this paper and future works.

The authors are with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464-8603, Japan, e-mail: {yutaka,honda,hiro}@ertl.jp

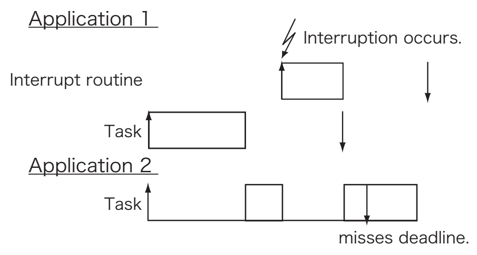


Fig. 1. Deadline miss by execution of interrupt routines prior to tasks in a global EDF scheduling.

II. Hierarchical Scheduling

A. System Model

The system is modeled with N applications, denoted by $A = \{A_1, A_2, \ldots, A_n\}$. Each application A_i is characterized by tuple (U_i, D_i, IDT_i), where U_i is a utilization of CPU, D_i is a minimum relative deadline and IDT_i is a maximum interrupt disabled time. Note that IDT_{ij} is a maximum interrupt disabled time in a dedicated CPU. A_i is a set of processing units, denoted by $\tau_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$. τ_{ij} must be either a task or an interrupt routine, characterized by a tuple $(p_{ij}, d_{ij}, idt_{ij})$, where p_{ij} is a static execution priority, d_{ij} is a relative deadline and idt_{ij} is a maximum interrupt disabled time on a dedicated CPU. In this paper, we assume followings.

- The system is closed and static real-time system where application characteristics are known in advance.
- Interrupt routines are assigned higher execution priority than all tasks in the same application.
- Total utilization $\sum_1^n U_i$ must be lower than 1.
- Each A_i is schedulable on a dedicated CPU with relative speed U_i and All applications are integrated on a shared CPU with relative speed 1.
- U_i, D_i and IDT_i for each application are given from the design specifications or the result of the dedicated verifications.

B. Hierarchical Scheduling Framework

Figure 2 presents a two-level hierarchical scheduling framework based on BSS algorithm. This framework consists of local schedulers, a global scheduler and two interfaces called Application Programming Interface (API) and Scheduler Programming Interface (SPI). A global scheduler schedules applications in EDF scheduling, and a local scheduler corresponding to each application schedules tasks in static priority-based scheduling, such as RM scheduling or DM scheduling. To improve isolation between local schedulers and the global scheduler, we introduce two modifications to BSS algorithm. One is that the location of residual list is moved from each scheduler to the global scheduler. In BSS algorithm, if each local scheduler takes misses in budget calculation, then the budget management conflicts. Another one is that the SPI is defined to make a clear border between local schedulers and the global scheduler. Table I shows a function list of the SPI. The local schedulers call the SPI functions to notice the results of local scheduling and request task

TABLE I
SCHEDULER PROGRAMMING INTERFACE.

Function Name	Parameter	Particularities
`set_deadline`	Absolute deadline	Notice absolute deadline of the application.
`set_schedtsk`	Task ID	Notice task ID that indicates scheduled task in the application.
`dispatch`	void	Request task dispatching with saving task context.
`exit_and_dispatch`	void	Request task dispatching without saving task context.

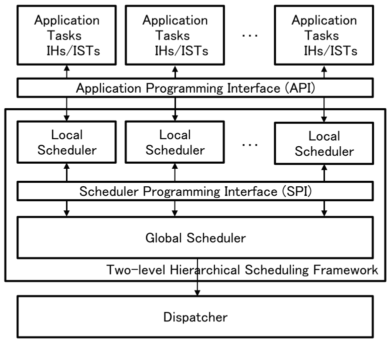

Application Tasks IHs/ISTs
Application Tasks IHs/ISTs
...
Application Tasks IHs/ISTs
Application Programming Interface (API)
Local Scheduler
Local Scheduler
...
Local Scheduler
Scheduler Programming Interface (SPI)
Global Scheduler
Two-level Hierarchical Scheduling Framework
Dispatcher

Fig. 2. Constitution of the hierarchical scheduler based on BSS algorithm.

dispatching. The global scheduler determines available budget of each application according to the noticed deadline and the residual list of the application.

C. Bandwidth Sharing Server

We briefly recall the budget management of BSS algorithm. A budget element in the residual list of Application A_i is denoted by tuple $l_{ij} = (d_{ij}, b_{ij})$, where d_{ij} is an absolute deadline, b_{ij} is the available budget until d_{ij}. The residual list is a set of budget elements and ordered by non-decreasing absolute deadline. Three operations are defined on the list: addition, update, and delete.

Addition. A new element l_{ij} is created and inserted in the residual list when the local scheduler of application A_i notices its application deadline d_{ij} and an element corresponding to d_{ij} is not exist in the residual list. The operation of addition is following procedure.

1) Find the position for the new element l_{ij}, that is:

$$\exists l_{i(k-1)}, l_{ik} \; d_{i(k-1)} < d_{ij} < d_{ik}$$

2) Calculate the budget available until d_{ij} by the following equation.

$$b_{ij} = \min\{(d_{ij} - t_c) * U_i, (d_{ij} - d_{i(j-1)}) * U_i + b_{i(k-1)}, b_{ik}\}$$

where U_i is the utilization of an application A_i and t_c is current time in the system.

3) Insert the l_{ij} between $l_{i(k-1)}$ and l_{ik} in the list.

Update. Every time the application leaves the CPU, the residual list must be updated. It could happen for any of following reasons:

- the processing unit has finished execution.
- the budget has been exhausted.
- the application has been preempted by another application with an earlier deadline.

Then, the algorithm picks the element in the list corresponding to the actual deadline of the application, say the $k-th$ element, and updates the budgets all l_{ij} in the following way:

- b_{ij} = b_{ij} - e (j $\geq$ k)
- l_{ij} is removed. (j $<$ k $\wedge$ $b_{ij} > b_{ik}$)

where e denotes execution time of the processing unit.

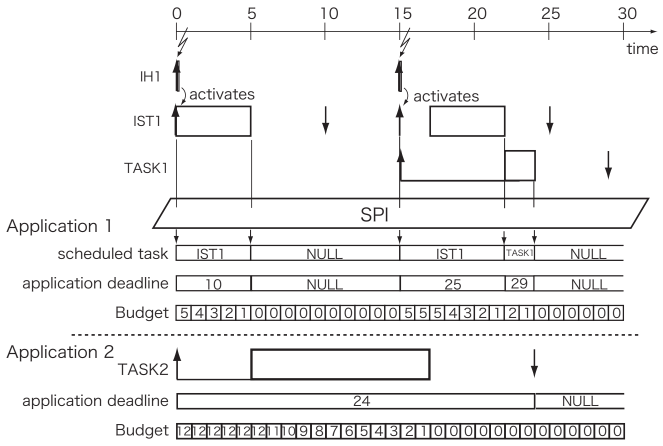


Fig. 3. Example of execution sequence in the hierarchical scheduling.

Delete. This operation is to delete an element that cannot contribute to the calculation of any new element. In this paper, we skip the details. The details of the delete operation and the correctness of BSS algorithm is described in [2].

III. INTERRUPT ROUTINE SCHEDULING

A. Interrupt Routine Model

In this section, we present a split interrupt routine model for a two-level hierarchical scheduling. In this model, the interrupt routine is split to an IH, and an IST. An IH provides device-depended service (e.g. clear an interrupt request) and activate an IST. On the other hands, an IST provides application-specified service corresponding to the interrupt source. In this paper, we assume that execution time of IH is negligible instant and handle an IST as a preemptive task with same priority as the corresponding interrupt routine. Figure 3 presents an example of execution sequence of two applications with the interrupt routine model. At $t = 0$, an interrupt occurs, then IH1 is executed and activates IST1. Because absolute deadline 10 of IST1 becomes earliest absolute deadline in the application 1, the local scheduler calls `set_deadline(10)` and `set_schedtsk(IST1)`. At the same time, TASK2 is activated, and the local scheduler of application 2 notices its application deadline 24 through the SPI. At this instant, deadline of application 1 (10) is earlier than deadline of application 2 (24). Therefore the global scheduler executes IST1 immediately. At $t = 5$ IST1 finishes, then the local scheduler updates application deadline and scheduled task. Since no tasks are ready in application 1, the global scheduler schedules TASK2 of application 2. At $t = 15$, the interrupt occurs and IST1 is activated again, then the deadline of application 1 becomes 25. In this situation, the deadline of application 2 (24) is earlier than deadline of application 1 (25). Therefore TASK2 continues. IST1 is delayed until TASK2 finishes.

B. Implementation results

In order to evaluate overhead, we implement the prototype system introduced the interrupt model on an actual processor. The prototype

TABLE II
PROCESSING TIME OF BASIC FUNCTIONS IN THE PROTOTYPE SYSTEM.

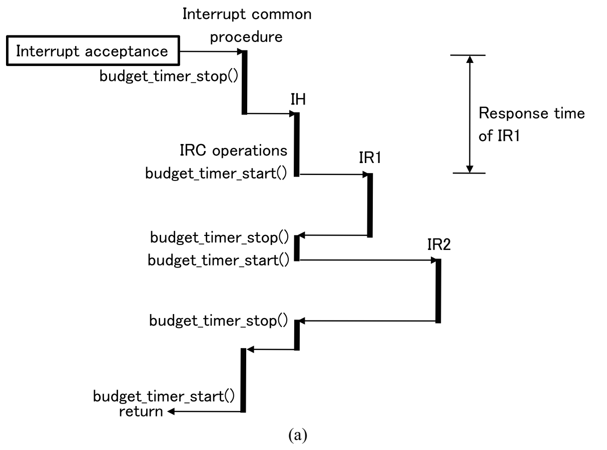
Instruction	Function	Processing time (us)
`act_tsk`	Activate task with task switching	62
`iact_tsk`	Activate task in kernel mode	4
`budget_timer_start`	Start the budget timer	6
`budget_timer_stop`	Stop the budget timer	7
`set_schedtsk`	Notice scheduled task ID (SPI)	5
`set_deadline`	Notice an application deadline (SPI)	1

TABLE III
RESPONSE TIME OF INTERRUPT ROUTINES.

Interrupt Routines	Response Time (us)
Interrupt Handler	3
Normal Interrupt Routine	10
Interrupt Service Task	37

system is developed based on TOPPERS/ASP Kernel which is almost compliant with the μITRON4.0 specification standard profile [12], and an open source real-time kernel distributed in [13]. The target system is the OAKS32R (oaks electric, Inc) with M32R processor (Frequency;66MHz, cache mode;OFF) which has been used in automotive control systems. In this evaluation, all executive objects including data and code section are allocated to on-chip SDRAM. Code size of TOPPERS/ASP Kernel for OAKS32R is 37KB which is only kernel code excluding application code. On the other hand, the code size of prototype system is 50KB. Since introducing the hierarchical scheduler causes most of 13KB, the effects of introducing the interrupt routine model is a little. In TOPPERS/ASP Kernel, an ISR is executed using the common kernel stack. However, because an IST requires specific stack like a normal task, data size of an application will increase.

The processing time of basic functions in the prototype system are shown in Table. II. Table. III presents the average response time of each interrupt routines obtained through the measurements for 10000 times. Here the response time is defined as the elapsed time from the instant when interrupts are accepted until corresponding interrupt routines are activated. Since an IST must be waited until an interrupt handler finishes and the execution switches to the IST, its response time gets longer by 27us compared to a conventional interrupt routine described in Figure. 4. In Most of real-time applications, the overhead introduced by the proposed interrupt routine model could be allowed because the overhead introduced by ISTs is small compared to activating task with task switching (`act_tsk()`).



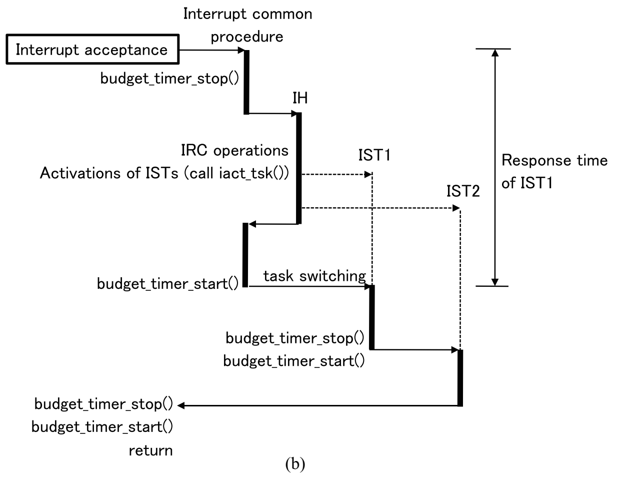


Fig. 4. Interrupt processing sequences in (a) conventional interrupt model (an IR means a normal interrupt routine) and (b) the proposed interrupt routine model.

C. Schedulability Analysis

We present schedulability analysis for real-time applications with interrupt disabled time. Firstly, we confirm that all tasks of an application are schedulable in a shared CPU. Based on the results of [2], the following theorem is obtained.

Theorem 1: If all ISTs and normal tasks of an application are schedulable in a dedicated CPU with relative speed U_i, then they are schedulable in a shared CPU with relative speed 1 by BSS algorithm.

Proof: ISTs are handled as tasks with higher priority than all tasks in the same application. In the view of the global scheduler, each application seems a sporadic task with deadline D_i. Therefore, if $U_i / \sum_{j=1}^{N} U_j \leq 1$ and all ISTs and normal tasks of an application A_i is schedulable in the dedicated CPU with relative speed U_i, then all applications are schedulable in the shared CPU with relative speed 1 by BSS algorithm. ■

Theorem 1 also means that an application budget is not exhausted during execution of ISTs and tasks. Therefore, if a task in an application disables interrupt, then budgets of other applications are not exhausted during each execution. Secondly, we analyze the influence of interrupt disabled time among applications and give necessary and sufficient condition of schedulability. This condition is derived based on [7].

Theorem 2: If an application A_i with interrupt disabled time satisfies following condition, the application is schedulable in a shared CPU with relative speed 1.

$$\forall d_i \in [0, \max_i\{D_i\}],$$
$$d_i \geq \max_{i<j}\{IDT_j * U_j\} + \sum_{D_k \leq d_i} (d_i * U_k) \quad (1)$$

Proof: This is easily derived from Theorem 12 in [7]. $\max_{i<j}\{IDT_j * U_j\}$ is blocked time by interrupt disabled time of applications with longer relative deadline than A_i. $\sum_{D_k \le d_i}(d_i * U_k)$ is summation of blocked time by applications with earlier absolute deadline than d_i and execution time of A_i itself. ■

To calculate by only D_i and IDT_i of each application, we delete parameters related to such the worst case execution time and relative deadline of each task from the condition (1), then the following equation is obtained.

$$\begin{aligned} d_i &\ge \frac{\max_{i<j}\{IDT_j * U_j\}}{1-\sum_{D_k \le d_i}(U_k)} \\ &\ge \frac{\max_{i<j}\{IDT_j * U_j\}}{U_j} \\ &\ge \max_{i<j}\{\frac{IDT_j * U_j}{U_j}\} \\ &\ge \max_{i<j}\{IDT_j\} \\ &\ge \max_{i}\{IDT_i\} \end{aligned}$$

After left side becomes minimum, we obtain the following sufficient condition.

$$\min_i\{D_i\} \ge \max_i\{IDT_i\} \quad (2)$$

To illustrate the use of the proposed schedulability test method, we utilize two simple examples. In example 1, an applications set is given with $\{(0.25, 6, 1), (0.25, 6, 1), (0.5, 8, 7)\}$. In this case, the minimum relative deadline is $\min\{6, 6, 8\} = 6$ and the maximum interrupt disabled time is $\max\{1, 1, 7\} = 7$, then the condition (2) is not satisfied. Therefore an application in the applications set may miss deadlines. In example 2, an applications set is given with $\{(0.25, 6, 2), (0.25, 6, 2), (0.5, 20, 4)\}$. Then the minimum relative deadline is $\min\{6, 6, 20\} = 6$ and the maximum interrupt disabled time is $\max\{2, 2, 4\} = 4$. In this case, the condition (2) is satisfied. Therefore the applications set is schedulable in the shared CPU. Although the proposed schedulability test condition is sufficient but not necessary, it is valuable to the system integrators in the system design phase because they can determine quickly which applications could be integrated to the system even without detail knowledge of each application.

IV. Related Works

Many hierarchical scheduling have been proposed. The Open System [1] guarantees that utilization of CPU on a shared CPU corresponds with a dedicated CPU. G. Lipari proposed BSS algorithm [2] and PShED algorithm [3] that can guarantee that all tasks satisfy the timing constraints in a shared CPU. Since BSS and PShED require only deadlines of tasks, the algorithms are applicable compared to the Open System [1]. However the prior works assumed that applications consist of only tasks, and ignored the existence of interrupt routines. J. Regehr proposed and implemented a general hierarchical scheduling framework described in [5], [6]. They used a fixed priority scheduler as the root scheduler to schedule interrupt routines prior to all tasks in [6]. Therefore each application needs to be resolved into tasks and interrupt routines for mapping proper execution environments. On the other hand, we use an EDF scheduler as the root scheduler to respectively schedule each application that includes interrupt routines without reconstructions of applications. Moreover, this paper presents response time of an interrupt routines and proposes schedulability test method for integrated applications without characteristics of each task.

Several interrupt routine models are presented in [9], [10]. Although they targeted for a conventional real-time systems or a general-purpose system, not a hierarchical scheduling system. In this paper, we introduce the split interrupt routine model to a two-level hierarchical scheduling system. [11] presents schedulability analysis with interrupt routines. However it assumed that interrupt occurs periodically, and an interrupt disabled time was not mentioned. Having interrupt disabled time in tasks is similar to having critical sections in tasks where they access global resources that are shared between applications. [8] presents a schedulability analysis of global resources sharing in EDF+SRP (Stack Resource Policy). It assumes that WCET parameters are given to calculate demand bound function in verification phase. Although these schedulability analyses are useful to check the schedulability in detail, however, even if design or source code of a task is changed only a little, characteristics of the task and the schedulability analysis of all applications need to be calculated again. In practice, if each application is independently developed in a different company, system integrators could not obtain detail knowledge of each task in the system design phase.

V. Conclusion

This paper presents a split interrupt routine model for a two-level hierarchical scheduling. By using this model, the main part of an interrupt routine is handled as a task and become controllable by a global EDF scheduler. Moreover we analyze the schedulability of the applications and give a simple feasibility test method. As a future works, we will refine the prototype system to improve the response time an ISTs and evaluate it with real applications for a real automotive control system.

References

[1] Z. Deng and J.W.-S. Liu and L. Zhang and S. Mouna and A. Frei, *An Open Environment for Real-Time Applications*, Real-Time Systems Journal, 1999.

[2] G. Lipari and K.Baruah, *Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems*, In Proc. of IEEE Real-Time Technology and Applications Symposium, 2000.

[3] G. Lipari and J. Carpenter and S. Baruah, *A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments*, In Proc. of IEEE Real-time System Symposium, November 2000.

[4] I. Shin and I. Lee, *Compositional Real-time Scheduling Framework*, In Proc. of IEEE Real-time Systems Symposium, December 2004.

[5] J. Regehr and A. Stankovic, *HLS: A Framework for Composing Soft Real-Time Schedulers*, In Proc. of the 22nd IEEE Real-time Systems Symposium, December 2001.

[6] J. Regehr and A. Reid and K. Webb and M. Parker and J. Lepreau, *Evolving Real-Time Systems using Hierarchical Scheduling and Concurrency Analysis*, In Proc. of the 24th IEEE Real-Time Systems Symposium, December 2003.

[7] L. George and N. Rivierre and M. Spuri, *Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling*, INRIA RECQUEN-COURT, September 1996.

[8] N. Fisher, M. Bertogna, and S. Baruah, *Resource-locking durations in edf-scheduled systems*, In Proc. of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium, April 2007.

[9] L. E. Leyva-del-Foyo and P. Mejia-Alvarez and D. de Niz, *Predictable Interrupt Scheduling with Low Overhead for Real-Time Kernels*, In Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing System and Applications, August 2006.

[10] Y. Zhang and R. West, *Process-Aware Interrupt Scheduling and Accounting*, In Proc. of the 27th IEEE Real-Time Systems Symposium, December 2006.

[11] K. Jeffay and D. L. Stone, *Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems*, In Proc. of the 14th IEEE Real-Time Systems Symposium, December 1993.

[12] ITRON, *μITRON 4.0 Specification Ver.4.03.00*, 2007.

[13] TOPPERS Project. http://www.toppers.jp/