International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:11, 2012

Customization of a Real-Time Operating System Scheduler with Aspect-Oriented Programming

Kazuki Abe, Myungryun Yoo and Takanori Yokoyama

Abstract—Tasks of an application program of an embedded system are managed by the scheduler of a real-time operating system (RTOS). Most RTOSs adopt just fixed priority scheduling, which is not optimal in all cases. Some applications require earliest deadline first (EDF) scheduling, which is an optimal scheduling algorithm. In order to develop an efficient real-time embedded system, the scheduling algorithm of the RTOS should be selectable. The paper presents a method to customize the scheduler using aspect-oriented programming. We define aspects to replace the fixed priority scheduling mechanism of an OSEK OS with an EDF scheduling mechanism. By using the aspects, we can customize the scheduler without modifying the original source code. We have applied the aspects to an OSEK OS and get a customized operating system with EDF scheduling. The evaluation results show that the overhead of aspect-oriented programming is small enough.

Keywords—aspect-oriented programming, embedded system, operating system, real-time system

I. Introduction

An application program of a real-time embedded system is usually designed as a set of tasks. For example, an automotive engine control application program in an electronic control unit (ECU) consists of a number of tasks for fuel injection, ignition, emission control and diagnosis. A real-time operating system (RTOS) is used to manage tasks of a real-time embedded system. For example, OSEK OS [1] is widely used in automotive control systems. Most RTOSs, including OSEK OS, adopt fixed priority scheduling. However, fixed priority scheduling algorithms such as rate monotonic (RM) scheduling and deadline monotonic (DM) scheduling are not optimal in all cases.

Earliest deadline first (EDF) scheduling is an optimal scheduling algorithm [2][3], but there are few commercial RTOSs with EDF scheduling. Bimbard et al. have extended OSEK OS to support EDF scheduling and showed the benefits of the extended OSEK OS[4]. However, all applications do not require EDF scheduling. The overhead of EDF scheduling is larger than the overhead of fixed priority scheduling in general. The fixed priority scheduling is suitable for some applications and EDF scheduling is suitable for other applications. However, it is difficult for one RTOS to provide both the scheduling algorithm because the resources of embedded systems are limited. In order to develop an efficient real-time system, the scheduling algorithm of a RTOS should be statically selectable.

Diederichs et al. have presented an EDF scheduler plug-in, which is an application-level scheduler for OSEK OS[5]. The plug-in can be used for applications that require EDF scheduling. However, an application-level scheduler is not efficient. A customizable RTOS scheduler is required for better efficiency.

K. Abe, M. Yoo and T. Yokoyama are with Graduate School of Engineering, Tokyo City University, 1-28-1, Tamazutsumi, Setagaya-ku, Tokyo 158-8557 Japan, e-mail: g1181501@tcu.ac.jp, {yoo, yokoyama}@cs.tcu.ac.jp.

Aspect-oriented programming[6] has been applied to the customization of operating systems[7][8][9][10][11]. Those researches utilize aspect-oriented programming for the customization of several properties of operating systems, but not for the customization of the schedulers. Hotun et al. have presented a feature model for schedulers[12]. They also discuss the use of aspect-oriented programming for their purpose, but the implementation is their future work.

The paper presents a method to customize the scheduler of a RTOS using aspect-oriented programming. We replace the fixed priority scheduling mechanism of an OSEK-compliant operating system called TOPPERS/OSEK kernel[13] with an EDF scheduling mechanism using aspect-oriented programming. We define aspects for EDF scheduling in AspeCt-oriented C (ACC): C-based aspect-oriented programming language[14][15]. The code for EDF scheduling is woven into TOPPERS/OSEK kernel by the ACC compiler.

The rest of the paper is organized as follows. Section II describes scheduling mechanisms of the RTOS. Section III describes a customization method with aspect-oriented programming and presents aspects to customize the scheduler. We evaluate the performance of the customized operating system in Section IV. Section V concludes the paper.

II. Scheduler of Real Time Operating System

A. OSEK OS Scheduler

The scheduling algorithm of OSEK OS is fixed priority scheduling. The priority levels of tasks are statically declared in OIL (OSEK Implementation Language)[16] and cannot be dynamically changed. The declared priority levels of tasks are stored in configuration data, which are generated from the OIL description file by the system generator (SG). Periodic tasks are implemented with an alarm mechanism connected with a counter that is used for the system timer.

Fig. 1 shows the fixed priority scheduling mechanism of TOPPERS/OSEK kernel. There are prioritized ready queues, in which tasks with state *ready* are queued according to their priority levels. The priority levels of the tasks are stored in the task control block (TCB). When the running task is terminated or suspended, the scheduler selects the first task in the highest-priority no-empty ready queue to be executed by the CPU as shown in Fig. 1. The scheduler also works when a task is activated or released and a preemption occurs if the priority of

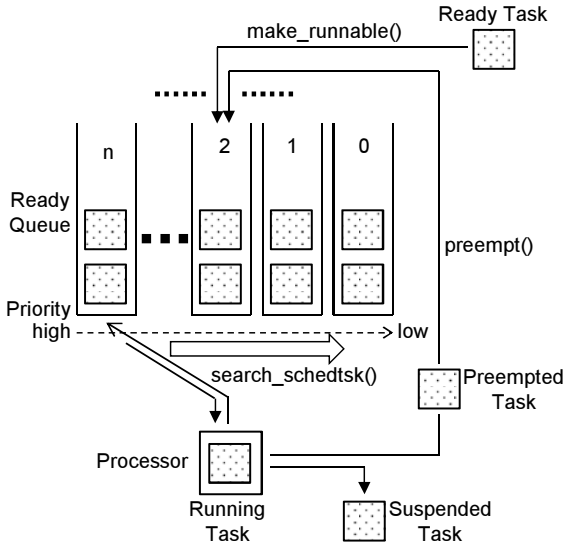

Fig. 1. Fixed Priority Scheduling Mechanism

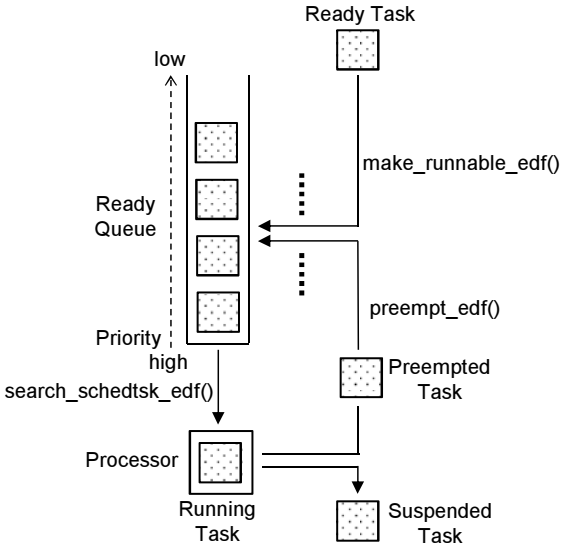

Fig. 2. EDF Scheduling Mechanism

the activated or released task is higher than the priority of the running task. The preempted task is enqueued in a prioritized ready queue according to the priority level.

B. EDF Scheduling Mechanism

We make the scheduling algorithm selectable: fixed-priority scheduling or EDF scheduling. Fig. 2 shows the mechanism of an EDF scheduling algorithm. Tasks are queued in a single ready queue according to their absolute deadlines. When the state of a task transits to *ready*, the task is inserted in the ready queue according to the absolute deadline. The task with the earliest absolute deadline other than the running task is to be at the head of the ready queue. When the running task is terminated or preempted, the scheduler selects the task at the head of the ready queue to be executed by the CPU. The preempted task is enqueued in the ready queue according to the absolute deadlines.

The absolute deadlines are stored in the TCB. The absolute deadline of a task is determined by adding the relative deadline to the time when the task is activated. To utilize EDF scheduling, the relative deadlines of tasks must be known. In OSEK OS, the attributes of tasks such as priorities must be defined in OIL. So we extend OIL to declare the relative deadlines of tasks and their values are stored in the configuration data. When a task is activated, the EDF scheduler calculates the absolute deadline of the task referring to the relative deadline in the configuration data. We are now extending the SG to automatically generate configuration data including the relative deadlines of tasks, and we manually add the relative deadlines of tasks to the configuration data referring to the extended OIL description.

If all tasks managed by the operating system are periodic tasks and their relative deadlines are equal to their periods, we don't have to declare the relative deadlines in OIL because the absolute deadlines can be calculated with the periods. So we also present aspects for EDF scheduling in the case that all tasks are periodic tasks. We don't have to declare the relative deadlines in OIL in this case.

III. Aspects for Customization

A. Customization with Aspect-Oriented Programming

We use ACC, C-based aspect-oriented programming language, to customize the scheduler of TOPPERS/OSEK kernel because TOPPERS/OSEK kernel is implemented in C language. ACC is an aspect-oriented programming language based on the join point model like AspectJ[17] and Aspect C++[18].

An aspect consists of pointcuts and advice. A pointcut represents a set of join points. Join points are identifiable points in the execution of a program. ACC supports join points such as *call*, *execution*, *set* and *get*. Advice is code to be executed at join points matched by a pointcut. ACC supports *before*, *after* and *around* advice. We can run the advice code before or after the join point by defining *before* advice or *after* advice. We can also run the advice code instead of the join point by defining *around* advice.

ACC is implemented as a translator, which inputs ACC source files and C source files and outputs C source files to be compiled by a C compiler. Fig. 3 shows a customization flow based on the ACC compilation process. Original C source files of TOPPERS/OSEK kernel and ACC and C source files for the customization are transformed to C source files of the customized operating system by ACC compiler (translator). Then the C source files are compiled by a C compiler.

B. Aspects for EDF Scheduling

We present aspects to replace the fixed priority scheduling mechanism illustrated by Fig. 1 with the EDF scheduling mechanism illustrated by Fig. 2. We show the aspects in the case that the relative deadlines are used to calculate the absolute deadlines in this section.

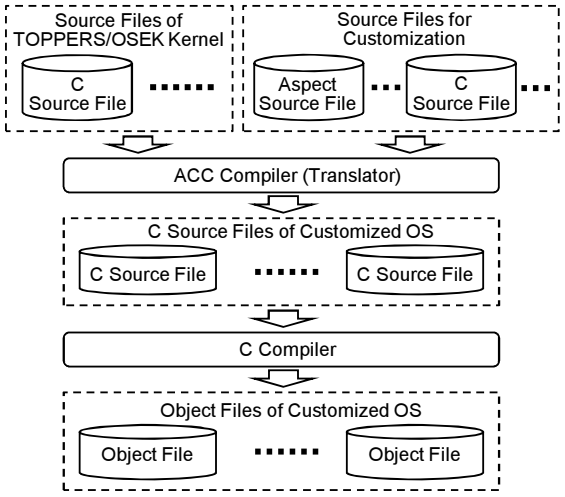

Fig. 3. Customization with ACC

```
/* replace fixed-priority-based task search
                        with EDF-based task search */
void around() :
    execution(void search_schedtsk(void)) {
  search_schedtsk_edf();
    /* get the task at the head of the ready queue
       and remove the head from the queue */
}
```

Fig. 4. Aspect to Extract a Task to be Executed

Fig. 4 shows an aspect to get a task to be executed by the CPU. The aspect replaces the task search for fixed priority scheduling with the task search for EDF scheduling. The around advice of the aspect calls function *search_schedtsk_edf()* instead of executing the code of function *search_schedtsk()* that selects the first task in the highest-priority no-empty ready queue as shown in Fig. 1. Function *search_schedtsk_edf()* selects the task at the head of the ready queue to be executed by the CPU as shown in Fig. 2.

Fig. 5 shows an aspect for the ready queue operation. When the state of a task transits to *ready*, the task is enqueued in the ready queue. The aspect replaces the prioritized ready queue operation for fixed priority scheduling with the deadline-based ready queue operation for EDF scheduling. The around advice of the aspect calls function *make_runnable_edf()* instead of executing the code of function *make_runnuble()* that enqueues the ready task in a prioritized ready queue as shown in Fig. 1. Function *make_runnable_edf()* enqueues the ready task in the ready queue as shown in Fig. 2, referring to the absolute deadlines of the tasks stored in the TCB.

Fig. 6 shows an aspect for preemption. The aspect replaces the preemption mechanism for fixed priority scheduling with the preemption mechanism for EDF scheduling. The around advice of the aspect calls function *preempt_edf()* instead of executing the code of function *preempt()* that enqueues a preempted task in a prioritized ready queue as shown in Fig. 1. Function *preempt_edf()* enqueues the preempted task in the ready queue according to the absolute deadlines as shown in

```
/* replace prioritized ready queue operation
        with deadline-based ready queue operation */
BOOL around(TaskType tskid) :
    execution(BOOL make_runnable(TaskType))
    && args(tskid) {
  return(make_runnable_edf(tskid));
    /* enqueue the task specified by argument tskid
       (insert the task into the ready queue
        accoording to the absolute deadlines) */
}
```

Fig. 5. Aspect for Deadline-Based Ready Queue Operation

```
/* replace fixed-priority-based preemption
                        with EDF-based preemption */
void around() :
    execution(void preempt()) {
  preempt_edf();
    /* enqueue the ruuning task
       and call function search_schedtsk_edf() */
}
```

Fig. 6. Aspect for EDF-Based Preemption

Fig. 2.

The absolute deadlines stored in the TCB must be maintained for EDF scheduling. Fig. 7 shows an aspect to call function *update_deadline()* before executing the code of function *ActivateTask()*. Function *update_deadline()* updates the absolute deadline of the activated task referring to the relative deadline in the configuration data. The value of the absolute deadline is calculated by adding the value of the relative deadline to the time when the task is activated.

Fig. 8 shows aspects for EDF scheduling other than the aspects shown above. The first aspect replaces the execution of the code of function *Schedule()* with the call of function *Schedule_edf()* that executes the EDF scheduler. The second aspect replaces the initialization of the fixed-priority scheduler with the initialization of the EDF scheduler. The third aspect replaces deleting the head of the prioritized ready queue for fixed priority scheduling with deleting the head of the ready queue for EDF scheduling.

C. Aspects for EDF Scheduling of Periodic Tasks

We show the aspects in the case that all tasks managed by the operating system are periodic tasks and their relative deadlines are equal to their periods in this section. In this case, we don't have to declare the relative deadlines in OIL because

```
/* update absolute deadline */
before(TaskType tskid) :
    execution(StatusType ActivateTask(TaskType))
    && args(tskid) {
  update_deadline(tskid);
    /* update the absolute deadline of the task
       specified argument tskid (activated task)
       referring to the relative deadline
       in the configuration data */
}
```

Fig. 7. Aspect for Absolute Deadline Update

```
/* replace fixed-priority scheduler call
                          with EDF scheduler call */
StatusType around() :
    execution(StatusType Schedule()) {
  return(Schedule_edf());
    /* call the EDF scheduler */
}

/* replace fixed-priority scheduler initialization
                 with EDF scheduler initialization */
void around() :
    execution(void task_initialize(void)) {
  task_initialize_edf();
    /* initialize the EDF scheduler */
}

/* replace prioritized queue head operation
        with deadline-based queue head operation */
TaskType around() :
    execution(Inline TaskType ready_delete_first($)) {
  return( ready_delete_first_edf() );
    /* delete the head of the ready queue */
}
```

Fig. 8. Other Aspects for EDF Scheduling

```
/* update absolute deadline */
after(AlarmType almid) :
    call(void enqueue_alarm(AlarmType))
    && args(almid) {
  update_deadline_alm(almid);
    /* update the absolute deadline of the task
       activated by the alarm specified by argument
       almid */
}
```

Fig. 9. Aspect for Absolute Deadline Update with Alarm

the absolute deadlines of tasks can be calculated with the periods not relative deadlines. However, the periods of tasks are not explicitly declared. Periodic tasks are implemented by the alarm mechanism of OSEK OS. The absolute deadlines can be obtained by referring to the parameters of the alarm mechanism because the absolute deadline of a periodic task is equal to the next activation time of the task. The aspects shown below are used in this case instead of the aspect shown by Fig. 7.

Fig. 9 shows an aspect to update the absolute deadline of a periodic task. The after advice of the aspect calls function *update_deadline_alm()* after the call of function *enque_alarm()* that enqueues the alarm. Function *update_deadline_alm()* sets the absolute deadline in the TCB referring to the next activation time of the alarm specified by argument *almid*.

Fig. 10 shows an aspect for initialization of EDF scheduler. The before advice of the aspect calls function *deadline_initialize()* before starting the dispatcher. Function *deadline_initialize()* sets up the alarm-id to task-id table and initialize the absolute deadlines of the TCB. The alarm-id to task-id table is used to get the identifier of the task activated by the alarm specified by the alarm identifier in function *update_deadline_alm()*.

```
/* initialize absolute deadlines */
before() : call(void start_dispatch() {
  deadline_initialize();
    /* set up the alarm-id to task-id table, and
       initialize the absolute deadlines of TCB */
}
```

Fig. 10. Aspect for Absolute Deadline Initialization with Alarm

TABLE I
EXECUTION TIME OF *ActivateTask()*

Operating System (Scheduling)	*ActivateTask()* Execution Time [μsec]	
	without task switch	with task switch
Customized with Aspects (EDF)	49.20	50.55
Directly Rewritten (EDF)	45.65	47.30
Original (Fixed Priority)	25.40	26.50

IV. EXPERIMENTAL EVALUATION

A. Experimental Environment

We have done experiments to evaluate the performance of the customized operating system with EDF scheduling. We use an evaluation board with M16C/26 microprocessor with a clock rate of 20MHz. The microprocessor contains 64kB flash ROM and 2kB RAM. The base operating system is TOPPERS/OSEK kernel ver.1.1. We also use ACC Compiler ver.0.9, GCC ver.4.2.2, C compiler NC30 ver.5.10, assembler AS30 ver.4.20, linker LN30 ver.410 and load module compiler LMC ver.3.30.00.

We have measured the execution times of system calls and the memory consumption of the operating system customized with the aspects shown in section III-B. We have also directly rewritten the source code of TOPPERS/OSEK kernel to support EDF scheduling and measured the execution times of system calls and the memory consumption for comparison.

B. Experimental Results

Table I shows the execution time of system call *ActivateTask()*. We have measured the execution time in two cases: the case without a task switch and the case with a task switch. The number of tasks is 10 in the every case. The table shows execution times of three operating systems: the operating system customized with aspects, the directly rewritten operating system with EDF scheduling and the original TOPPERS/OSEK kernel with fixed priority scheduling. In EDF scheduling, the execution time increases by 3.25 μsec a task, because the ready queue is scanned when a task is activated.

The difference between the execution time of the operating system customized with aspects and the execution time of the directly rewritten operating system means the overhead of aspect-oriented programming. The overhead is less than 10% of the execution time in the worst case, so we think the overhead is sufficiently small.

Table II shows the memory consumption of the operating systems. The table shows the size of the code section, the size of the data section of ROM and the size of the data section of RAM. The memory consumption of the data section depends

TABLE II
MEMORY CONSUMPTION

Operating System (Scheduling)	Memory Consumption [Byte]		
	Code	Data(ROM)	Data(RAM)
Customized with Aspects (EDF)	9625	64	425
Directly Rewritten (EDF)	9535	64	425
Original (Fixed Priority)	9085	74	418

on the number of tasks. The table shows the value in the case that no application task exists. In EDF scheduling, the size of the data section of ROM increases by 5 bytes a task and the size of the data section of RAM increases by 5 bytes a task. In original TOPPERS/OSEK kernel, the size of the data section of ROM increases by 7 bytes a task and the size of the data section of RAM increases by 5 bytes a task.

The difference between the code size of the operating system customized with aspects and the code size of the directly rewritten operating system is about 1% and negligible. We think the customization with aspect-oriented programming is efficient enough.

V. CONCLUSION

We have presented a method to customize the scheduler of a RTOS using aspect-oriented programming in this paper. We have defined aspects to replace the fixed priority scheduling mechanism of an OSEK OS with an EDF scheduling mechanism without modifying the original source code. We have applied the aspects to TOPPERS OSEK kernel and have got a customized operating system with EDF scheduling. We have also evaluated the customized operating system and have got the result that the overhead of aspect-oriented programming is small enough. We are going to present aspects to customize other mechanisms of the RTOS, for example, the resource mechanism with the priority-ceiling protocol.

ACKNOWLEDGMENT

K. Abe, M. Yoo and T. Yokoyama thank the developers of TOPPERS/OSEK kernel and the developers of AspeCt-oriented C. This work is supported in part by KAKENHI (24500046).

REFERENCES

[1] OSEK/VDX, *Operating System, Version 2.2.3*, 2005.
[2] Liu, C. L. and Layland, J. W., Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol.20, No.1, pp.46–61, 1973.
[3] Dertouzos, M. L., Control Robotics: The Procedural Control of Physical Processes, *Proceedings of IFIP Congress 1974*, pp.807–813, 1974.
[4] Bimbard, F. and George, L., EDF Feasibility Conditions with Kernel Overheads on an Event Driven OSEK System, *Proceedings of third International Conference on Systems*, pp.277–284
[5] Diederichs, C., Margull, U., Slomka, F. and Wirrer, G., An application-based EDF scheduler for OSEK/VDX, *Proceedings of Design, Automation and Test in Europe 2008*, pp.1045–1050, 2008.
[6] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. Loingtier, J. M. and Irwin, J., Aspect-Oriented Programming, *Proceedings of the 11th European Conference on Object-Oriented Programming*, pp. 220–242, 1997.
[7] Beuche, D., Fröhlich, A. A., Reinhard, M., Papajewski, H., Schön, F., Schröder-Preikschat, W., Spinczyk, O. and Spinczyk, U., On Architecture Transparency in Operating Systems, *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating system*, pp. 147–152, 2000.
[8] Coady, Y., Kiczales, G., Feeley, M. and Smolyn G., Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code, *Proceedings of the 8th European Software Engineering Conference*, pp.88–98, 2001.
[9] Afonso, F., Silva, C., Montenegro, S. and Tavares, A., Applying Aspects to a Real-Time Embedded Operating System, *Proceedings of the 6th Workshop on Aspects, Components, and Patterns for Infrastructure Software*, Article No.1, 2007.
[10] Park, J. and Hong S., Building a Customizable Embedded Operating System with Fine-Grained Joinpoints Using the AOX Programming Environment. *Proceedings of the 2009 ACM symposium on Applied Computing*, pp.1952-1956, 2009.
[11] Lohmann, D., Hofer, W., Schröder-Preikschat, W. and Spinczyk, O., Aspect-Aware Operating System Development, *Proceedings of the 10th International Conference on Aspect-Oriented Software Development 2011*, pp.69-80, 2011.
[12] Hatun, K., Bockisch, C., Sözer, H. and Akşit, M., A Feature Model and Development Approach for Schedulers, *Proceedings of the 1st Workshop on Modularity in Systems Software*, pp.1–5, 2011
[13] TOPPERS Project, http://www.toppers.jp/en/
[14] Gong, M., Zhang, C. and Jacobsen, H.-A., Systems Development with AspeCt-oriented C (ACC), *Connections 2007 (ECE Graduate Symposium, University of Toronto)*, Talk 5.6, 2007.
[15] AspeCt-oriented C, https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/
[16] OSEK VDX, *OSEK/VDX System Generation OIL: OSEK Implementation Language Version 2.5*, 2004.
[17] Kiczales, G., Hilsdale, E., Hugonin, J, Kersten, M., Palm, J. and Griswold, W. G., An Overview of AspectJ, *Proceedings of the 15th European Conference on Object-Oriented Programming*, pp.327–353, 2001.
[18] Spinczyk, O., Gal, A., and Schröder-Preikschat, W., Aspect C++: An Aspect-Oriented Extension to the C++ Programming Language, *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems*, pp.53–59, 2002.