Novel Control Method for Preventing Missed Deadlines in Periodic Scheduling

Yuuki Furukawa, Toshihiro Yamauchi and Hideo Taniguchi
Graduate School of Natural Science and Technology
Okayama University
Okayama, Japan
furukawa@swlab.cs.okayama-u.ac.jp, {yamauchi, tani}@cs.okayama-u.ac.jp

***Abstract*—Processing that is executed periodically must be completed before the next release time. If such processing is not completed before the next release time, the processing that had been scheduled is not executed. This is complicated by the fact that the execution time from release to the end of periodically executed processing is not constant, due to changing I/O processing time and the influence of timer interrupts. To solve this, we propose a system that records the execution time of the processing, judges whether the processing will be finished before the specified deadline, and can execute appropriate processing that can be completed within the remaining time. In this paper, we describe the design and evaluation of our system.**

Keywords-deadline-miss; periodic scheduling; control overhead; real-time system

I. INTRODUCTION

Processings that control the motors and sensors of robots are executed periodically. These periodically executed processings must be completed before the next release time. Many real-time scheduling algorithms have been proposed to realize periodic scheduling [1]–[4]. However, there is no method that ensures that the processing does not miss its completion deadline. If periodically executed processing is not completed before its specified deadline; the processing scheduled for execution in the next period is not executed, causing a problem. For example, in existing methods, missed deadline is ignored and processing continues, or processing is forced termination. Obviously, if a processing that misses a specified deadline is executed continually, then that processing cannot be said to be executing exactly periodically. Conversely, forcing a processing that misses a specified deadline to terminate, adversely affects processings related to that processing.

On the other hand, the execution time of periodically executed processing from release to the end is not constant due to changing I/O processing time and the influence of timer interrupts. In embedded systems such as robots [5], the processing periods are shortened so as to realize highly precise control, and the influence of the dispersion for the execution time is large. In addition, the processing tends to be developed highly [6]. Therefore, there are many requests for functional expansions. Hence, it is essential that a developer knows precisely time when a processing will be finished and the release time of the next period.

Existing systems uses a worst-case execution time to confirm whether a periodic processing satisfy deadlines. There are analysis methods to predict worst-case execution time. The result values of worst-case execution time prediction are pessimistic. Therefore, available execution time for the periodic processing is shortened. In addition, the existing methods need knowledge and experience for the periodic processing, and the cost of the developer is high.

Therefore, the proposed system records the processing execution time, judges whether the processing will be completed before the specified deadline, and can execute appropriate processing that will be completed within the remaining time. The proposed system can prevent missed deadlines in periodic scheduling. Hence, the proposed method reduces the limitation of the periodic processing execution time. Moreover, the proposed method can support the creation of the periodic processing for the developer. In this paper, we describe the design and evaluation of our system.

II. PERIOD EXCESS PREVENTION METHOD

The proposed method (called the period excess prevention method) judges whether processing will be finished before the time when the processing must be finished (the end time), and executes processing appropriate for the remaining time (emergency processing) if it finds that the originally scheduled processing will not be completed on time. The period excess prevention method has the following characteristics:

(1) Periodic processing is divided into multiple modules, and the state of the execution for that processing is monitored.
(2) The execution time of the modules is saved.
(3) Emergency processing that will be completed within the remaining time is executed if the system judges that the originally scheduled processing will not be finished before the end time.

The underlying basic mechanism of our period excess prevention method is depicted in Fig. 1. We assume a user mode process has a periodic processing. An operating system (kernel) controls the periodic execution of the process. The period excess prevention method divides periodic processing into multiple modules, and monitors the state of execution for that processing. A module is part of a divided periodic processing and is executed only once per period. At the time

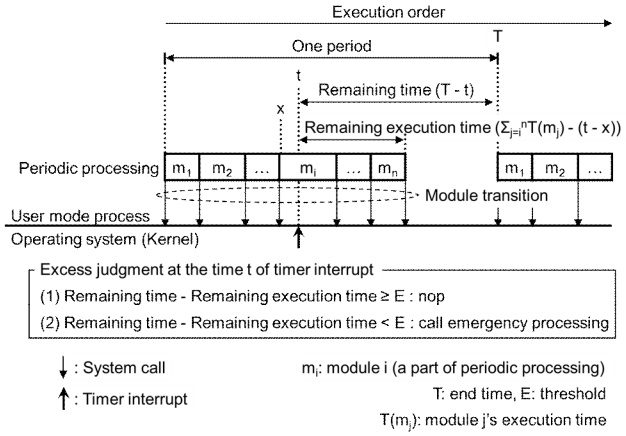

Figure 1. Basic mechanism of proposed method

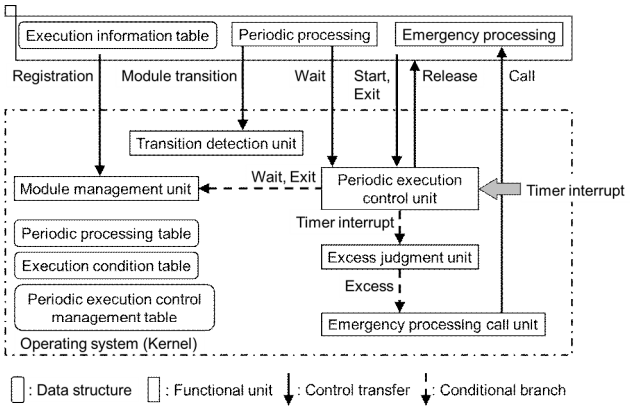

Figure 2. Basic structure of the proposed system

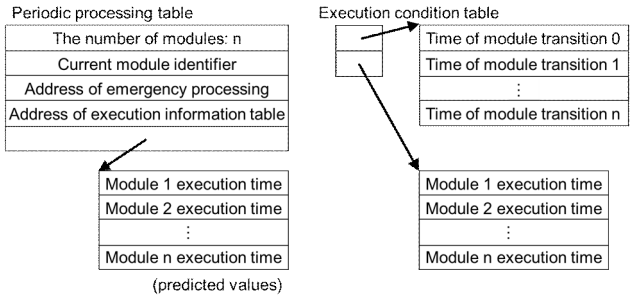

Figure 3. Details for the periodic processing table and execution condition table

of program creation, the developer divides the periodic processing to any modules with optional length. As each module is completed and the next module executed (module transition), periodic processing invokes a system call and sends that information of the module transition to the operating system. The operating system records the time of module transition, computes the execution time for each module based on this module transition time, and then saves it in memory.

When timer interrupts occur, the system compares the time remaining from the current time to the end time (remaining time) and the remaining periodic processing execution time (remaining execution time), and judges whether the processing will be finished before the specified end time. The remaining time is the difference in time between the time t of the timer interrupt and the end time T. The remaining execution time is the value obtained by subtracting from the total execution time $\Sigma_{j=i}^{n}T(m_j)$ of the module that has not finished to the time from the start time x of the current module m_i to time t. If the value of the difference in time between the remaining time and the remaining execution time is greater than or equal to threshold, the system determines that the processing will be completed before the end time. Otherwise, the system determines that the processing will exceed the deadline and calls the emergency processing. In addition, threshold E is used to avoid missing the deadline due to dispersion of module execution time and control overhead.

III. DESIGN

A. Design Policy

We will now explain the design policy of our period excess prevention method.

(1) Function division between process and operating system.

In the proposed method, the part of the processing that depends on the process is processed by the process in order to minimize the interaction between processes and operating system and to reduce control overhead.

(2) Separation of procedure division and management data division.

By separating procedure division and management data division, the functional expansion for the execution of multiple processes is facilitated.

B. Basic structure

The basic structure of the proposed system is depicted in Fig. 2, and explained below. In the process, periodic processing and emergency processing comprise a procedure division, while an execution information table comprises a management data division. The operating system has a transition detection unit, a module management unit, an excess judgment unit, an emergency processing call unit, and a periodic execution control unit all comprising the procedure division. In addition, it has a periodic processing table, an execution condition table, and a periodic execution control management table that together constitute the management data division. The details for the periodic processing table and the execution condition table are shown in Fig. 3.

Periodic processing is executed periodically by the periodic execution control unit, and sends information of the module transition to the operating system. In addition, when processing in one period is finished, periodic processing sends the requirement of a wait to the periodic execution control unit.

Emergency processing is requested by the emergency processing call unit when it is judged that the periodic processing will not finish before the end time.

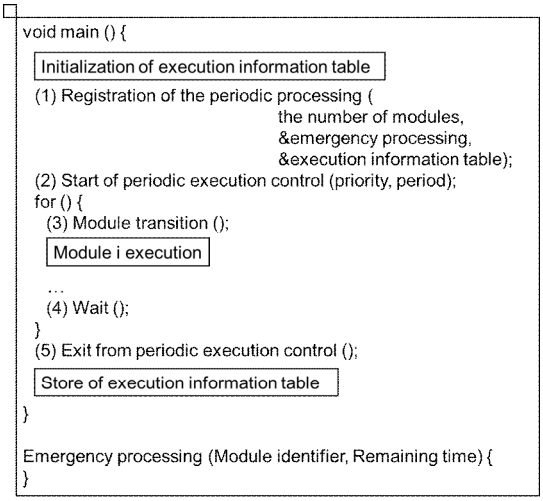

Figure 4. Processing flow

The execution information table has the module execution time for each module in the periodic processing.

The transition detection unit updates the current module identifier in the periodic processing table to the next module identifier when it detects module transition. In addition, the transition detection unit records the time of module transition in the execution condition table.

The module management unit manages the registration and unregistration of the periodic processing. In addition, when the periodic processing for one period is finished, the module management unit calculates the module execution times and records those in the periodic processing table.

The excess judgment unit compares the remaining time and the remaining execution time at the timer interrupt, and judges whether the periodic processing will be finished before the end time.

The emergency processing call unit calls the emergency processing if the excess judgment unit determines that the processing will not finish before the end time.

The periodic execution control unit periodically releases the processes that have periodic processing.

The periodic processing table manages the information necessary to determine if excess periodic processing will occur; while the execution condition table manages the execution condition of the periodic processing in the recent period; and the periodic execution control management table manages the information necessary to execute the processes periodically.

If a process has a periodic processing table and an execution condition table, the interaction between processes and operating system decreases at the time of module transition. However, the control overhead becomes large at the timer interrupt because it is necessary for the operating system to refer to a periodic processing table and an execution condition table in a process. Therefore, an operating system has the periodic processing table and the execution condition table in Fig. 2. In addition, an operating system has functional units in Fig. 2 to reduce the control overheads.

TABLE I. INTERFACE

	Function	Interface
1	Registration of periodic processing	register(modulenum, rfunc, infomt); modulenum: the number of modules rfunc: address of emergency processing infomt: address of execution information table
2	Start of periodic execution control	enter(prio, usec); prio: priority usec: period
3	Module transition	next();
4	Wait	wait();
5	Exit from periodic execution control	exit();

C. *Processing flow*

The processing flow of the process for the system is outlined in Fig. 4, while the interface is shown in Tab. 1. As outlined in Fig. 4, the processing of the process involves the initialization and storing of the execution information table, the registration of periodic processing, the sending of module transition information, and the starting, waiting, and exiting of periodic execution control. In addition, at the timer interrupt, the operating system judges the excess periodic processing. We will look at each stage of the process and the processing of the timer interrupt below.

Initialization of the execution information table is achieved by a process that records the ideal value or the past execution value for module execution time in the execution information table. Moreover, the execution information table is stored in order to utilize the past module execution time for proposed method. Before the process exits the run, the module execution time of the execution information table is saved to a file, where it becomes non-volatile.

Registration of periodic processing requires that the periodic processing information be registered in the periodic processing table. When registration of periodic processing is required, control is transferred to the module management unit. The module management unit then registers the number of modules used for the periodic processing, the address of emergency processing, and the address of the execution information table in the periodic processing table. In addition, calculating the data size from the number of modules, the module management unit allocates data area to store the module execution times for the periodic processing table and execution condition table, as well as the module transition time; further, it copies the module execution time in the execution information table to the periodic processing table.

The start of periodic execution control starts periodic execution control of the process. When periodic execution control is required to start, the periodic execution control unit registers the priority and the period of the process to the periodic execution control management table. Consequently, this process is released periodically by the periodic execution control unit.

In the module transition processing, transition information is sent to the operating system. When this information is sent, the transition detection unit gets the

```
void main () {
    read(fd, size, infomt);
    register(modulenum, rfunc, infomt);
    enter(priority, usec);
    for(;;) {
        next();
        main1();
        next();
        main2();
        wait();
    }
    exit();
    write(fd, size, infomt);
}
void rfunc (moduleid, time) {
    switch (moduleid) {
        case 0:
        if (time < TIME1) return func1();
        if (time < TIME2) return func2();
        ...
}
unsigned int infomt[MODULE_NUM];
```

Figure 5. Example for description of programs

current time, records it in the execution condition table, then updates the current module identifier in the periodic processing table to the next module identifier.

In the wait processing, information for the end of one period processing is sent to the operating system. When wait is required, control is transferred to the periodic execution control unit. The periodic execution control unit changes the process to the wait state until its next release time, and control is transferred to the module management unit. The module management unit gets the time of module transition from the execution condition table, calculates module execution times, and stores them in the execution condition table. Following this, predicted values for module execute times in the next period are calculated and stored in the periodic processing table. In addition, these predicted values are copied to the execution information table. Let us assume that the predicted value of the x-times module execute time is $T0_x$ while its actual value is T_x, the predicted values in the next period is calculated using the following formula:

$$T0_{x+1} = T0_x + y(T_x - T0_x) \text{ (y: any value)} \quad (1)$$

In the exit processing, the periodic processing exits periodic execution control of the process. When exit is required, the periodic control unit deletes the priority and the period of the periodic processing from the periodic control table, and control is transferred to the module management unit. The module management unit initializes periodic processing table and the execution condition table, and frees the data area that was allocated at the time of registration.

At the timer interrupt, the periodic execution control unit refers to the periodic execution control table, and releases all processes scheduled to be released at the same time if there are processes scheduled to be released at the timer interrupt, then control is transferred to the excess judgment unit.

The excess judgment unit gets the end time from the periodic execution control table and calculates the time remaining. In addition, the excess judgment unit gets the predicted values of the module execution times from the periodic processing table and calculates the remaining execution time. Following this, the excess judgment unit compares the remaining time with the remaining execution time, and judges whether the periodic processing will exceed the end time. If the excess judgment unit determines that the processing will exceed the end time, control is transferred to the emergency processing call unit. The emergency processing call unit then gets the address of the emergency processing from the periodic processing table, and then requests emergency processing. The current module identifier and the remaining time are then sent to the process. The emergency processing executes the appropriate processing based on the current module and the time remaining. After emergency processing, the process is terminated.

D. Description of programs

An example program description is shown in Fig. 5. Before periodic processing begins, the process reads the file and initializes the execution information table. In addition, the process calls register() and sends the number of modules in the periodic processing, modulenum, the address of emergency processing, rfunc, and the address of the execution information table, infomt. To call enter(), the process is executed periodically with priority prio and period usec. In Fig. 5, main1() and main2() are modules. The periodic processing calls next() at the time of module transition, and wait() at the end of one periodic processing. When wait() is called, the execution information table is updated. When the periodic processing exits the periodic execution control, exit() is called to unregister the periodic processing that was registered by register(). If it is determined that the periodic processing will exceed the end time, the operating system requests emergency processing. In Fig. 5, moduleid represents the current module identifier, and time represents the remaining time. For example, if the current module identifier is 0 and the remaining time is 5, the operating system calls rfunc(0, 5). When rfunc() is called, the process chooses and executes the appropriate processing suitable for the current module identifier and the remaining time.

IV. EVALUATION

A. Point of view

Using the proposed method, the state of the periodic processing can be known, and processing completed before the deadline. However, in order for the state of the periodic processing to be known, the periodic processing invokes a system call and sends module transition information. In addition, it is necessary for the operating system to calculate module execution times and to judge the excess periodic processing at the timer interrupt. Consequently, control overheads occur in the following three areas of processing:

(1) Module transition

In the proposed method, the periodic processing calls the system in order to send module transition information. Therefore, the module transition processing time is from the system call time to the time the system call returns.

TABLE II. MEASUREMENT ENVIRONMENT

CPU	Intel Pentium II 400MHz
Memory	96 MB
Timer interrupt period	1ms
Connection	None

TABLE III. CONTROL OVERHEADS (NUMBER OF MODULES = 10)

Processing content	Processing time (clocks)
Module transition	
overall system call	429 (1.07 μs)
transition detection unit	132 (0.33 μs)
Wait	
overall system call	2011 (5.03 μs)
module management unit	1747 (4.37 μs)
Timer interrupt	
overall timer interrupt	1507 (3.77 μs)
excess judgement unit	190 (0.48 μs)
Emergency processing call	296 (0.74 μs)

(2) Wait

When the periodic processing for one period is finished, a wait is sent. Then, in the proposed method, n module execution times are calculated, and the predicted value of each module execution time is calculated. In addition, these predicted values for the module execution times are copied to the execution information table.

(3) Timer interrupt

For periodic execution control, at the timer interrupt, the operating system judges whether there are processes scheduled to be released, and releases those processes. In the proposed method, after the above processing is finished, the remaining time and the remaining execution time are computed, and it is judged whether the periodic processing will exceed the end time.

We measured the time taken by the above three processings, and analyzed it in relation to control overhead. The measurement environment is shown in Tab. 2. The rdtsc instruction was used to record the time.

B. Control overheads

We measured the processing times of module transition, wait, and timer interrupt for 10 modules in a periodic processing. The measurement results are shown in Tab. 3. The result for the transition detection unit for module transition is the value obtained by subtracting call processing and system call return time from the module transition processing time. The result for the module management unit is the processing time from confirmation for the registration of periodic processing to initialization of the module identifier. The processing time for the timer interrupt in Tab. 3 is the result for the case when emergency processing is not called. The result for the excess judgment unit is the processing time taken to judge whether the processing will exceed the end time. In addition, the result for the emergency processing is the value from the time just before emergency call processing to the time immediately after call return. In this measurement, the emergency processing executes no operation.

For module transition, the processing time for the transition detection unit was approximately 30% of the processing time for the overall system call, and the control overhead for the system call is large. Because the total time for module transition in one period is dependent on the number of modules, the control overhead is large if there are a large number of modules.

The processing time for the wait is longer than the result for the module transition unit and excess judgment unit. This is because the processing of the wait includes the following: calculation of the module execution time, storing of the data in the periodic processing table and the execution condition table, and copying of the module execution time from the periodic processing table to the execution information table. In addition, we varied the number of modules used in the range from 1 to 10, and measured the processing time in each case. From this result, we found that the processing time for the module management unit can be given by the following formula:

$$370 + 150(n - 1) \quad (n: \text{the number of modules}) \qquad (2)$$

In addition, the processing time for system calls and changing of the process to a wait state is approximately 300 clocks and is constant.

For the timer interrupt, the processing time for the excess judgment unit was approximately 12% of the processing time for the entire timer interrupt. The excess judgment unit adds the execution time of the modules that did not finish at the timer interrupt. Therefore, the processing time increases if there are a large number of modules that are not finished. However, the dispersion for the processing time for the above calculation is small, and we confirmed that the dispersion of the processing time for the excess judgment unit was approximately 70 clocks for number of modules in the range 1-10.

C. Discussions

1) Ratio of the processing time for one period

We discuss the processing times of module transition unit, and the transition detection unit and excess judgment unit for one period.

The number of times module transition is called is equal to the number of modules for periodic processing in one period. In Tab. 3, the processing time for one module transition is 1.07 μs. In other words, if the number of modules is 1 and the period for the periodic processing is 1 ms, then the processing time for module transition would account for approximately 0.1%. However, the ratio for the processing time for module transition increases if the period is short and there is a large number of modules. For example, if the number of modules is 100 and the period for periodic processing is 1 ms, then the processing time for module transition will account for approximately 10%. Therefore, we can decrease the time taken to process the modules if we divide the periodic processing into a very small number of modules at the time of periodic processing division. The periodic processing division is discussed in section 4.C.2.

The module management unit calculates all module execution times when wait is required. Therefore, the periodic processing of the module management unit is proportional to the number of modules. In Tab. 3, the processing time of the module management unit is 4.37 μs if the number of modules is 10, and it is approximately 0.5% of the total if the periodic processing period is 1 ms. As in the module transition, the ratio for the module management unit processing time increases if the period is short and there is a large number of modules. For example, if the number of modules is 100 and the periodic processing period is 1 ms, then the module management unit processing time will account for approximately 3.8% of the total. Therefore, as in the module transition, it is necessary to take the number of modules into consideration when dividing the periodic processing.

The excess judgment unit is executed at every timer interrupt. In periodic execution control, one period of periodic processing is an integral (N) multiple of the period of timer interrupt. Therefore, the processing time ratio of one excess judgment for the period of a timer interrupt is equal to that of N times the excess judgment for the period of periodic processing. In Tab. 3, the periodic processing of the excess judgment unit is 0.48 μs, and accounts for approximately 0.05% of the 1 ms period for the timer interrupt. Because the period for motor control of a robot is in the range 10 ms to 100 μs, if the periodic processing period is 100 μs, it is approximately 0.5% even if the processing time ratio for the excess judgment unit is large, and it is small.

2) Division of periodic processing

We discuss the division of the periodic processing in this section.

The number of modules influences the total processing time for module transition and the processing time for the wait. In addition, the control overhead is proportional to the number of modules. Furthermore, module execution time influences the error value of the remaining execution time (the error value of the remaining execution time easily increases if the dispersion of the module execution time is large). The error in the remaining execution time is the value of the difference between the computed value for the remaining execution time and the actual remaining execution time. This is because the remaining execution time is computed by subtracting the time from the start time of the current module to the current time from the total execution time of the module that has not yet finished. Therefore, if the dispersion of the execution time for the periodic processing is large, the precision of the excess judgment becomes high due to an increase in the number of modules. By increasing the number of modules to shorten the execution time per module, the error in the execution time remaining becomes small. In addition, the dispersion of the execution time is small and the execution time for periodic processing is constant; by reducing the number of modules, the control overhead becomes small. In other words, by combining the modules in which the dispersion of the execution time is small and dividing the periodic processing in which the dispersion of the execution time is large into small modules with short execution times, the control overhead becomes small and the excess judgment becomes more precise.

3) Change factors for the execution time

If the dispersion of the execution time for the periodic processing is large and these change factors accumulate, the processing is more likely to exceed the deadline. The following are considered change factors:

(1) I/O processing time

(2) Interrupt processing (timer, end of I/O processing, etc.)

V. CONCLUSION

In this paper, we explained the basic underlying mechanism of our period excess prevention method and presented the design policy. In addition, we described its basic structure and interface. We also explained the flow of each process.

In the proposed method, the periodic processing invokes the system call at the time of every module transition. In addition, the operating system records module execution times, and judges whether the periodic processing will exceed the deadline at each timer interrupt. We evaluated the control overhead for one period and showed that the processing time of the excess judgment unit is small.

In addition, we discussed the division of periodic processing and change factors for the execution time.

Our future work will involve the clarification of the change factors.

ACKNOWLEDGMENT

This research was partially supported by Grant-in-Aid for Scientific Research 24300008.

REFERENCES

[1] C. Liu, J, Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," Journal of the ACM, Vol.20, pp.46-61, 1973.

[2] G. C. Buttazzo, "Rate monotonic vs. EDF: judgment day," Real-Time Systems, The International Journal of Time-Critical Computing, Vol.29, Issue 1, pp.5-26, 2005.

[3] S. Cho, S. Lee, A. Han, K. Lin, "Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems," IEICE Transactions on Communications, Vol.E85-B, No.12, pp.2859-2867, 2002.

[4] H. Cho, B. Ravindran, E. D. Jensen, "An Optimal Real-Time Scheduling Algorithm for Multiprocessors," Proceedings of the 27th IEEE International Real-Time Systems Symposium, pp.101-110, 2006.

[5] K. Yokoi, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara, H. Hirukawa, "Experimental Study of Humanoid Robot HRP-1S," International Journal Robotics Research, Vol.23, No.4-5, pp.351-362, 2004.

[6] G. Buttazzo, "Research trends in real-time computing for embedded systems," ACM SIGBED Review, Vol.3, Issue 3 pp.1-10, 2006.