International Journal of Control and Automation
Vol. 4, No. 2, June, 2011

Implementation and Evaluation for Sophisticated Periodic Execution Control in Embedded Systems

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

Abstract

Embedded systems are complicated, and a large number of these systems are widely used today. In embedded systems, the types of processings to be executed are limited, and many processes are executed periodically. In such systems, we need to reduce the overhead of periodic execution control and the dispersion of the processing time. ART-Linux has been proposed as a conventional real-time operating system that can be used for this purpose in various devices such as robots. In this paper, we discuss the periodic execution control of ART-Linux and clarify several problems. Next, we propose a design for sophisticated periodic execution control in order to solve these problems. In addition, we discuss the realization of periodic execution control, and the effects of the proposed control. Finally, we show the results of our evaluation for the release and the wait to clarify the effects of the proposed control.

Keywords: *Periodic execution control, Scheduling, ART-Linux, Control overhead, Real-time process*

1. Introduction

Embedded systems are complicated, and a large number of these systems are widely used today [1]. Therefore, it is important to upgrade the software that controls the resources of the embedded devices. In embedded systems, the types of processings to be executed are limited, and many processes are executed periodically. An example of such a process is a process that controls the motor of a robot; this process needs to be executed every 5 ms. Such a process must be executed before a deadline, and this is an example of real-time processing. In this case, we need to reduce the overhead of the periodic execution control and the dispersion of the processing time.

ART-Linux [2][3][4] has been proposed as a conventional real-time operating system that can be used in devices such as robots [5]. In ART-Linux, we can use the application program (AP) and the device driver for Linux, the real-time process with the short period can be periodically executed without significant jitter. There have been studies related to the improvement of the performance of ART-Linux [6].

In this paper, we discuss the periodic execution control of ART-Linux and clarify several problems. Next, we propose a design of a sophisticated periodic execution control to solve these problems. In addition, we discuss the realization of periodic execution control, and the effects of the proposed control. Finally, we present the evaluations of the control overhead and the dispersion of the processing time to clarify the effects of the proposed control.

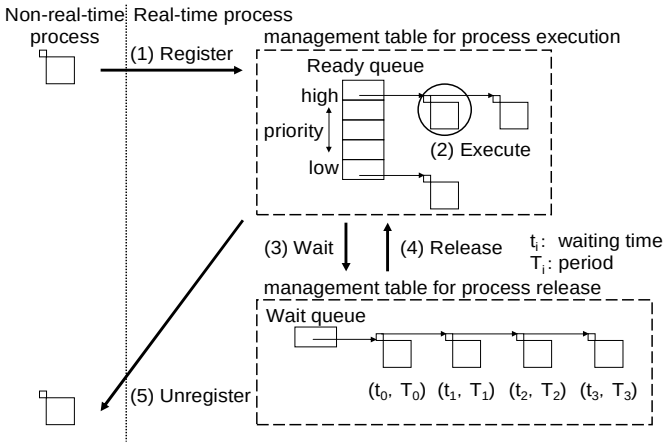

Figure 1. Periodic execution control of ART-Linux

2. ART-Linux

2.1. Periodic execution control

ART-Linux is a real-time operating system based on Linux. We refer to processes executed periodically as real-time processes; all other processes are termed non-real-time processes.

ART-Linux manages real-time processes and non-real-time processes separately. Real-time processes have a higher priority than non-real-time processes. That is, non-real-time processes are executed only when there are no real-time processes to be executed.

We show the periodic execution control of ART-Linux in figure 1. The ready queue in figure 1 manages the real-time processes that are in a ready state so that the processes can be executed on the basis of priority. We call this the ready state. The wait queue manages the real-time processes that are in a wait state. We call the transition from the wait queue to the ready queue a release. In the next few paragraphs, we explain the processing of the periodic execution control of ART-Linux.

1. Register
   When a non-real-time process requires registration, this process is connected to the ready queue. Consequently, it is managed as a real-time process. This real-time process is assigned a period and a priority.
2. Execute
   If there is already a real-time process in the ready queue, the real-time processes are executed on the basis of their priorities.
3. Wait
   A real-time process requires a wait when the processing of the real-time process for one period is finished. Then, this real-time process is disconnected from the ready queue, connected to the wait queue, and set a waiting time. The waiting time is the time from the next release time of the real-time process before this one in the wait queue to the next release time of this real-time process. In this way, the real-time process waits until its next release time.
4. Release
   At the time of a timer interrupt, the waiting time of the real-time process at the top of the wait queue is calculated. If the resulting value is less than 0, the real-time process is disconnected from the wait queue and connected to the ready queue. Then, the waiting

Table 1. Measurement environment

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

time of the next real-time process in the wait queue is calculated, and the release of this real-time process is judged. The above step is repeated until the waiting time after calculation is greater than 0 or until all real-time processes are disconnected from the wait queue. In other words, the above step is repeated until all real-time processes reaching a release time are connected to the ready queue.

5. Unregister
   When a real-time process requires unregistration, this process is disconnected from the ready queue and managed as a non-real-time process. Then, its period and priority are initialized.

2.2. Performance

2.2.1. Measurement environment and items: We measured the processing time from the timer interrupt to the execution of a real-time process to clarify the periodic execution control overhead in ART-Linux. Further, we measured the interval error of the periodic execution to clarify the precision of the periodic execution control. An interval of the periodic execution is the time from a processing start time to the next processing start time for a real-time process. In addition, the interval error is the difference in value between an interval and the period of a real-time process. We list the measurement environment in table 1. We used the rdtsc instruction to record the time.

2.2.2. Time from timer interrupt to execution of real-time process: We registered one real-time process with a period of 1 ms and the maximum priority and measured the time from the timer interrupt to the execution of this process 100 times. This result is shown in table 2, and the distribution of the time is shown in figure 2.

The difference between the average values and the maximum values is considerably larger than the difference between the average values and the minimum values in both table 2 and figure 2. We speculate that the dispersion of the time from the timer interrupt to the execution of the real-time process is large. One reason for this may be the fact that the processing time for the release is proportional to the number of real-time processes to be released at the same time, and the time from the timer interrupt to the execution of the real-time process increases. Therefore, we believe that the precision of the periodic execution will be low.

2.2.3. Measurement of interval of periodic execution: We registered one real-time process with a maximum priority and measured the interval of periodic execution. We set the period of the real-time process to be 1, 2, 4, 5, or 10 ms and measured the interval of periodic execution in each case. The interval error is shown in figure 3; we have explained it below.

The interval error of the periodic execution is proportional to the period shown in figure 3. We measured the precision of the timer interrupt period to clarify this factor. As a result of the measurement, we found that the actual timer interrupt interval is approximately 0.996 ms for the timer interrupt interval of 1 ms. Therefore, there is approximately 0.4% error for the

Table 2. Time from timer interrupt to execution of real-time process

Maximum (μs)	5.47
Average (μs)	2.84
Minimum (μs)	2.57
Dispersion (μs^2)	0.12

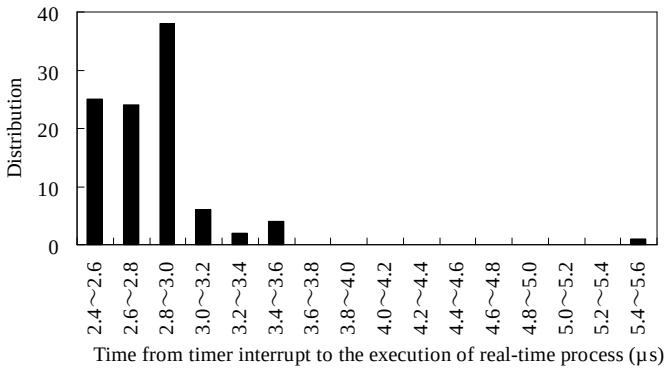

Figure 2. Distribution of time from timer interrupt to execution of real-time process

timer interrupt interval. In figure 3, the interval error of the periodic execution is approximately 4 μs for a period of 1 ms. That is, this error is affected by the error of the timer interrupt interval. In addition, we speculate that the timer interrupt does not occur by a correct period because of the difference between the timer interrupt period that the system requires and the precision of the timer movement clock.

2.3. Problems

We will discuss five problems to be clarified by the processing of the periodic execution control and the result of the measurement in ART-Linux.

1. The control overhead for a release is proportional to the number of real-time processes scheduled to be released at the same time, and the dispersion of the processing time for the release is considerable.
   For the release at the time of the timer interrupt in ART-Linux, the waiting time for a real-time process at the top of the wait queue is calculated, and the release of this process is judged. Then, all real-time processes scheduled to be released at this time are disconnected from the wait queue and connected to the ready queue. Therefore, the control overhead for the release is proportional to the number of real-time processes scheduled to be released at the same time. Therefore, if there are a considerable number of real-time processes scheduled to be released at the time of the timer interrupt, the processing time for the release will be significant. Further, when the dispersion of the number of real-time processes scheduled to be released at each time is considerable, the dispersion of the processing time is significant.
2. The control overhead of the connection processing to the wait queue for the wait is proportional to the number of real-time processes, and the dispersion of the processing time for the wait is considerable.

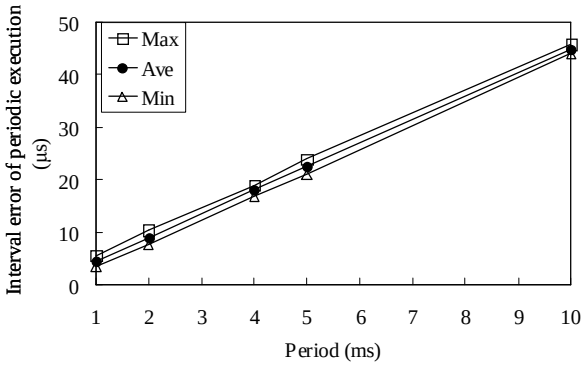

Figure 3. Interval error of periodic execution (ART-Linux)

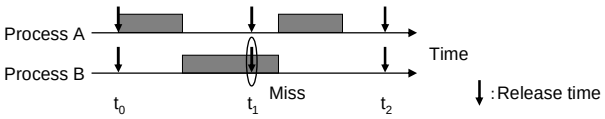

Figure 4. Example of real-time process that is not completely processed before next release time

For the wait of a real-time process in ART-Linux, the wait queue is searched from the top to calculate the waiting time of a real-time process that requires a wait and decide a position to insert this process into the wait queue. Therefore, processing that is proportional to the length of the wait queue is required; the control overhead for the wait is affected by the number of real-time processes and the waiting time of each real-time process in the wait queue. For example, if there are a considerable number of real-time processes that are released earlier than the real-time process that requires a wait, the processing time for the wait is significant. Hence, the processing time for the wait is long, and the dispersion of this processing time is considerable.

3. A real-time process that is not completely processed before the next release time has an adverse effect on the other real-time processes.
   Let us consider a case in which a real-time process is not completely processed before the next release time of a periodic execution control. Such a real-time process changes the interval of the periodic execution of the other real-time processes, and the precision of the periodic execution becomes low. In figure 4, process B is an example of such a real-time process. When process B requires a wait in ART-Linux, it waits from this time to the next release time. In this case, process B waits until the release time of t_2 not t_1. In other words, even if process B is not executed periodically on the basis of its period, the execution of this process continues periodically. Process B uses the time when process A is feasible, and the interval of the periodic execution of process A changes.
4. The dispersion of time from the timer interrupt to the execution of a real-time process is significant.
   As shown in Section 2.2.2, there is a case in which the change in the time from the timer interrupt to the execution of a real-time process is considerable. If such a change in time is big, the precision of the periodic execution is low. One of this reason is that the dispersion of the processing time for the release is significant.
5. A difference in the clock precision between a timer and a processor is not coordinated.

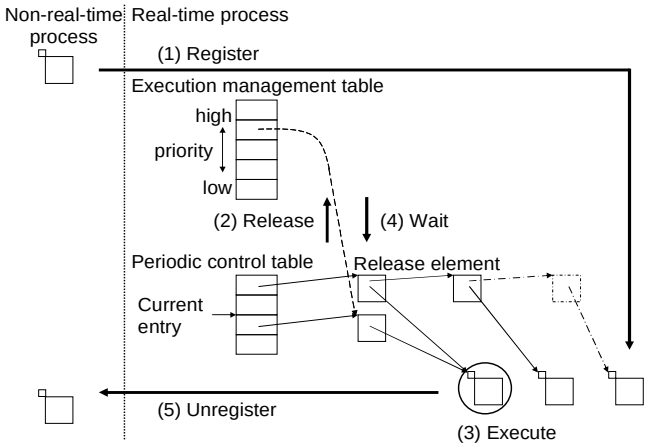

Figure 5. Sophisticated periodic execution control

As shown in Section 2.2.3, the interval error of the periodic execution is proportional to the period and increases because there is a difference in the clock precision between a timer and a processor. If the difference between the period and the interval of the period execution is considerable, the precision of the periodic execution will be low.

3. Sophisticated periodic execution control

3.1. Design

The processing time of the release and the wait for the periodic execution control of ART-Linux are affected by the number of real-time processes, and the dispersion of those processing time is significant. This results from the waiting time of a real-time process. Therefore, the sophisticated periodic execution control that we propose manages the real-time process scheduled to be released with respect to the time of each timer interrupt without managing the real-time processes that need to wait in one queue. Therefore, the proposed control can solve four of the five above-mentioned problems for the periodic execution control in ART-Linux.

In figure 5, we show the schematic representation of the sophisticated periodic execution control that can solve problems 1–4 of ART-Linux. The control can be explained as follows: The execution management table in figure 5 is equivalent to the ready queue shown in figure 1, and the periodic control table is equivalent to the wait queue.

The execution management table manages the real-time process to be executed. In particular, it manages the release element of the real-time process to be executed by connecting it to the entry of the execution management table on the basis of the priority of the real-time process. The release element is the element that has the information of the real-time process and has two queue entries. These entries are used for connecting the real-time process to the execution management table and the periodic control table. The periodic control table manages the real-time process scheduled to be released each time; the number of entries is equivalent to the longest period that can be set. One entry in this table is equivalent to one timer interrupt and is connected to the release element of the real-time process to be released at the timer interrupt. In other words, the number of entries in the periodic control table should be the least common multiple of the periods for all real-time processes.

We explain the processing of the sophisticated periodic execution control below.

1. Register
   When a non-real-time process requires registration, this process is managed as a real-time process by the periodic control table. Then, the periodic control table is searched, an entry that is connected to few release elements in the periodic control table is found, and the release element of the real-time process is connected to this entry. Simultaneously, the real-time process is set a period and a priority.
2. Release
   At the time of the timer interrupt, the current entry is translated to the next entry. The current entry is the entry that corresponds to the time of the timer interrupt in the periodic control table. If there is a release element in the current entry, the decision to release a real-time process is taken. In particular, the release element is connected to the execution management table on the basis of the priority of the real-time process. Then, the release element is not disconnected from the periodic control table. If the priority of the real-time process to be released is greater than that of the current process, a pre-emption occurs. In addition, if there is a release element of the real-time process to be executed in the execution management table at this time, all release elements of this real-time process are disconnected from the execution management table and the periodic control table, and this real-time process is terminated. This is done to restrain the adverse effect that this real-time process has on the other real-time processes.
3. Execute
   After the processing of release, wait, and unregister, if there is a release element in the execution management table, real-time processes are executed on the basis of their priorities. Then, the release element is not disconnected from the execution management table. Therefore, there is a release element of the real-time process being executed at the top of the entry corresponding to the priority of the execution management table. A bitmap is used for deciding which entry should be connected to the execution management table.
4. Wait
   When a real-time process requires a wait, its release element is disconnected from the periodic control table. Therefore, the real-time process waits until the next release time.
5. Unregister
   When a real-time process requires unregistration, all release elements of this process are disconnected from the periodic control table. Then, the period and the priority of this process are initialized. Hence, this process is managed as a non-real-time process.

3.2. Characteristics

The sophisticated periodic execution control that we proposed in Section 3.1 has the following characteristics:

1. The release of the real-time process is managed by the periodic control table.
2. A release element has two queue entries that are used for connecting the release element to the execution management table and the periodic control table.
3. A real-time process that is not completely processed before the next release time is terminated.
4. When the release element is connected, the dispersion for the number of release elements connected to each entry of the periodic control table becomes low.

According to characteristic 1, the calculation of the waiting time is unnecessary. Because of characteristic 2, the processing of the disconnection from the periodic control table for the

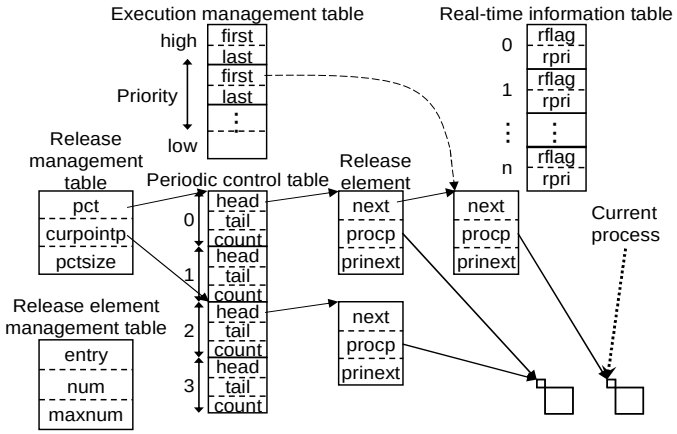

Figure 6. Data structure

release and the processing of the connection to the periodic control table for wait are unnecessary. Furthermore, because of characteristic 1 and 2, the number of queue operations required decreases, and the control overheads for the release and wait become small. In addition, we have speculated that the dispersion of the processing time for the release and wait become small because the number of processing for searching the release element is few. Because of characteristic 3, the influence of a real-time process that is not completely processed before the next release time on the other real-time processes is restrained. Because of characteristic 4, the dispersion of the processing time for the release becomes small.

3.3. Realization

3.3.1. Data structure: We have realized the sophisticated periodic execution control that we proposed earlier. We show the data structure of this periodic execution control in figure 6 and explain each management table below. The data structure of the periodic execution control has six management tables.

The release management table manages the information required to decide the release of real-time processes with the periodic control table. The current entry (curpointp) is translated into the next entry at every timer interrupt. Pct is the pointer to the top of the periodic control table, and pctsize is the number of entries in the periodic control table.

The number of entries in the periodic control table is the least common multiple of the periods of all real-time processes. Each entry manages the release element. All release elements that are connected with the entry to which curpointp points, (current entry) are connected to the execution control table for the release. The count of the periodic control table is the number of release elements to manage.

The release element contains information on the real-time process. Next is used for connecting the release element to the periodic control table, whereas prinext is used for connecting the release element to the execution management table. Procp is pointer to the real-time process.

The execution management table manages the release elements of the real-time processes that are to be executed on the basis of their priority. In this table, the release element of the real-time process being executed is at top of the entry that has the maximum priority among the entries that release elements are connected.

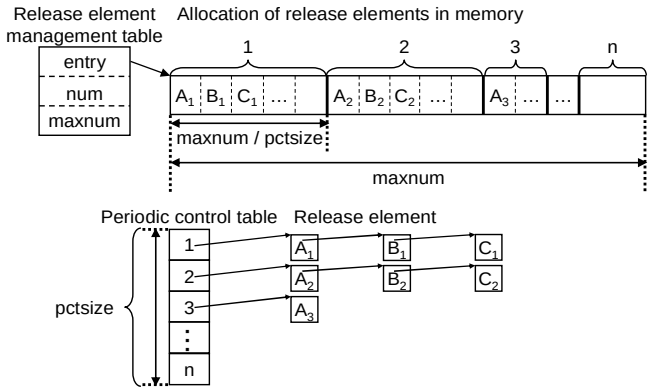

Figure 7. Allocation of release elements

The real-time information table manages the period and the priority of the real-time process. The number of each entry is equivalent to the process identifier. If the process management table manages the period and the priority of the real-time process, the access time may increase because the process management table has a considerable amount of information. Therefore, the cache miss is restrained, and the access time required to use the real-time information table becomes short. When a real-time process is registered, rflag is set a period and rpri is set a priority. If the real-time process is released, the most significant bit of rflag is set. Further, when the real-time process requires a wait, this bit is reset.

The release element management table manages the unused release elements. Num is the number of unused release elements, and maxnum is the number of all release elements that are allocated at the time of the initialization of all management tables.

3.3.2. Coding: Bear in mind the following for the implementation of the sophisticated periodic execution control.

First, the optimization of the code and the use efficiency improvement of the register used in the method are required in order to speed up processing. For example, we summarize the cords handling the same variable in one place in order to reduce the amount of rewriting in the register. As a result, the processing time becomes short.

Many release elements may be used by a real-time process in the proposed control. Therefore, the quantity of data to be referred to during one processing may become considerable, and the cache miss rate may become high. Moreover, the access time may increase significantly because the data spatial locality becomes low. Therefore, when the sophisticated periodic execution control is implemented, it is necessary to devise how to handle data such as the period and the priority of the real-time processes and how to allocate the release element.

Hence, we reduce the quantity of data for a real-time process and minimize the quantity of data for a release element. Further, during the processing of the release and the wait, data that do not relate with the periodic control are not read. The release element has only two queue entries and a pointer to the process. Information related to the real-time process, such as the period and the priority, is described in the real-time information table. This is done to avoid giving the information of the real-time process in the process management table and to reduce the quantity of data that is accessed. If the process management table contains the information of the real-time process, the quantity of data that needs to be accessed increases considerably because the process management table has a significant amount of information. Moreover, the

state of the real-time process as ready or wait is contained in rflag. We can use the same entry to manage the period and the state of the real-time process because the period of the real-time process is only changed in the processing of registration or unregistration. Therefore, we can reduce the quantity of data accessed, and the cache miss rate becomes low.

Figure 7 shows how to allocate release elements in the control that we have realized. At the initialization of each management table, release elements are allocated according to the maxnum size. Let x be the value that is obtained by dividing maxnum by pctsize. “i” is the serial number of each entry in the periodic control table. When the release element is connected to the i-th entry, the unused release element to use is searched from the (x * i)-th entry of all release elements. Therefore, the release element for each entry in the periodic control table is accumulated in the memory. Therefore, the data spatial locality is improved, and the access time for the release and wait decreases.

3.4. Effects

We have described the effects of the sophisticated periodic execution control below.

1. The control overhead of the release becomes small.
   For the release of a real-time process in ART-Linux, the waiting time for all real-time processes scheduled to be released at the same time is calculated; then, these processes are disconnected from the wait queue and connected to the ready queue. On the other hand, in the proposed control, the current entry is translated into the next entry, and all release elements of the current entry are connected to the execution management table. Therefore, it is unnecessary to calculate the waiting time and to disconnect the real-time processes from the wait queue. Moreover, the number of processing that is influenced by the number of real-time processes scheduled to be released at the same time decreases, and the dispersion of the processing time for release becomes small.
2. The control overhead of the wait becomes small, and the dispersion of the processing time for the wait becomes small.
   For the wait of a real-time process, in ART-Linux, the wait queue is searched from the top, and the waiting time for the real-time process that requires a wait is calculated. Then, the real-time process is disconnected from the ready queue and connected to the wait queue. On the other hand, in the proposed control, the release element is only disconnected from the execution management table. In addition, the release element of the real-time process being executed is at the top of the entry with its priority. Therefore, the processing time for the wait is short, and this value is fixed without affecting the number of the real-time processes in the wait queue and the waiting time of each real-time process.
3. The adverse effect of a real-time process that could not be completely processed before the next release time on the other real-time processes is restrained.
   In ART-Linux, the execution of the real-time process continues periodically even if the execution is not completed before the next release time. On the other hand, in the proposed control, if there is a release element of the real-time process still to be executed in the current entry as a release at the next release time, this real-time process is terminated. In other words, a real-time process that is not completely processed before the next release time is terminated. Therefore, the influence of such a real-time process on the other real-time processes is restrained.
4. The dispersion of the processing time for the release is small.
   In the proposed control, for registration, when the release element is connected to the periodic control table, the periodic control table is searched, an entry managing few

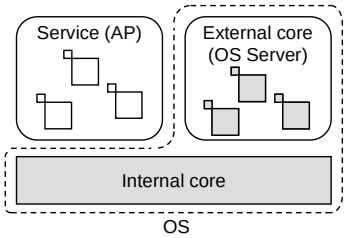

Figure 8. Basic architecture of *AnT*

release elements in the periodic control table is found, and the release element of the real-time process is connected to this entry. Therefore, the dispersion for the number of release elements being connected to each entry of the periodic control table becomes low. Hence, when there are real-time processes with different periods, the dispersion of the processing time for a release becomes small. As a result, the dispersion of the time from the timer interrupt to the execution of a real-time process becomes small.

4. Evaluation

4.1. *AnT* operating system

We have implemented and evaluated the proposed control in ***AnT*** [7][8]. ***AnT*** is an operating system based on microkernel architecture. The basic architecture of ***AnT*** is shown in figure 8. The program consists of OS and service. The OS comprises the kernel, which is called as the internal core, and the external core, which is executed as processes (OS server). Service consists of AP processes that execute the applications program. The internal core is the program part that guarantees the execution for a minimum number of common functions in all the systems. As the main function, the internal core has functions of scheduling management and memory management. The external core is the necessary program part to adapt to the use cases of the systems, it has a dynamically reconfigurable structure. For example, ***AnT*** offers the functions of input/output control, file management, and device driver as processes. Service is the program part that offers services.

AnT has the following characteristics: fast inter-server program communication mechanism, process control mechanism for dynamic running mode switch, adaptive-control function, and dynamic OS server replacement mechanism. The fast inter-server program communication mechanism offers high-speed communication control using copy-less data transfer, protection of the transfer data and provision of interface that adapted to the use case of the systems [8]. The dynamic running mode switch mechanism is mechanism that can switch the running mode of a process dynamically and freely. The adaptive-control function dynamically changes the software function and structure to adapt to the use case of the system. The dynamic OS server replacement mechanism allows us to continue the service without stopping the operating system when the OS server is extended or when the OS server is malfunctioning.

4.2. Evaluation items

We have clarified the effects of the proposed control by comparison of ***AnT*** and ART-Linux. We evaluated the following.

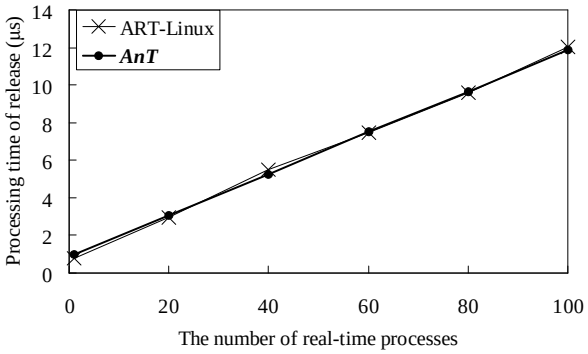

Figure 9. Processing time of release

1. The processing time of the release when there are a large number of real-time processes scheduled to be released at the same time
2. The processing time of the wait when there are a large number of real-time processes in the wait state
3. The dispersion for the number of real-time processes to be released at the same time

In periodic execution control, a real-time process that is not completely processed before the next release time has an adverse effect on the other real-time processes. In the proposed control, a real-time process that is not completely processed before the next release time is terminated. After this control is used, there is no influence of such a real-time process on the other real-time processes. Therefore, we have not evaluated quantitatively effect 3 in this study.

We evaluated the periodic execution control for ***AnT*** and ART-Linux in measurement environment of table 1. In addition, a non-real-time process that reads data of 1 Mbyte is executed to remove the influence of the CPU cache. The number of real-time processes registered in ***AnT*** is the same as those registered in ART-Linux.

4.3. The processing time of the release

To compare the control overhead of a release, we measured the processing time of the release when the number of real-time processes is N. In this manner, we evaluated effect 1. We registered N processes with a period of 1 ms and measured the processing time of the release at the time of each timer interrupt. In other words, we measured the processing time of the release when the number of real-time processes scheduled to be released at the same time was N. We changed the number of real-time processes in the range from 1 to 100 and measured the processing time in each case. The result is shown in figure 9.

In the figure, the processing time for ***AnT*** is found to equal to that for ART-Linux. In ART-Linux, for the release, the waiting time for all real-time processes scheduled to be released at the same time is calculated; then, these processes are disconnected from the wait queue and connected to the ready queue. On the other hand, in the proposed control, the current entry is translated into the next entry, and all the release elements of the current entry are connected to the execution management table. Therefore, it is unnecessary to calculate the waiting time and to disconnect the real-time processes from the wait queue. Hence, we speculated that the processing time of the release decreases. In addition, as there is a decrease in the number of the processings that are influenced by the number of real-time processes

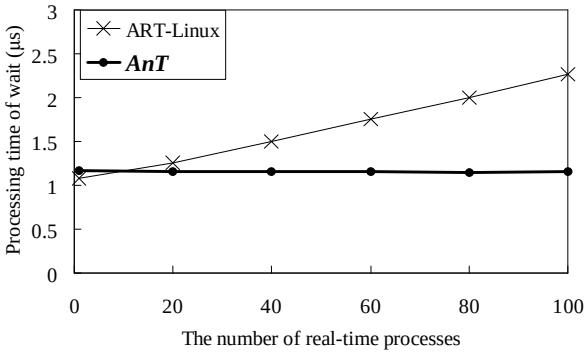

Figure 10. Processing time of wait

scheduled to be released at the same time, we speculated that the dispersion of the processing time for release becomes small. However, the results of ***AnT*** and ART-Linux do not show the difference in figure 9.

One of the reasons may be that the data spatial locality of ***AnT*** is lower than that of ART-Linux. In ART-Linux, only one data structure is used by a real-time process. On the other hand, in the proposed control, many data structures may be used by a real-time process. When the number of real-time processes is 1, the processing time of the release for ***AnT*** is longer than that for ART-Linux. In addition, when the number of real-time processes increases, the difference in the processing time between ***AnT*** and ART-Linux decreases, and there is a case in which the processing time of release for ***AnT*** is shorter than that for ART-Linux. Therefore, as the data spatial locality of ***AnT*** is lower than that of ART-Linux, we speculated that the effect of the proposed control is relatively low and the processing time of the release for ***AnT*** is equal to that for ART-Linux.

4.4. The processing time of the wait

To compare the control overhead of a wait, we measured the processing time of the wait when a real-time process required a wait. In this manner, we evaluated effect 2. We registered N real-time processes with a period of 1 ms and measured the processing time of the wait for the N-th real-time process after the timer interrupt. This result is the processing time of the wait for the case in which there are (N - 1) real-time processes that are released earlier than the real-time process that requires a wait. We changed the number of real-time processes in the range from 1 to 100 and measured the processing time in each case. The result is shown in figure 10.

As shown in the figure, the processing time for ***AnT*** is short and this value is fixed. The processing time for ART-Linux is proportional to the number of real-time processes shown in this figure. In ART-Linux, for the wait, the wait queue is searched from the top, and the waiting time for the real-time process that requires a wait is calculated. Then, the real-time process is disconnected from the ready queue and connected to the wait queue. Therefore, the processing time of the wait for ART-Linux is affected by the real-time processes in the wait queue. On the other hand, in the proposed control, the release element is only disconnected from the execution management table. Therefore, the value of the processing time for ***AnT*** is fixed and small, and it does not affect the number of real-time processes in the wait state.

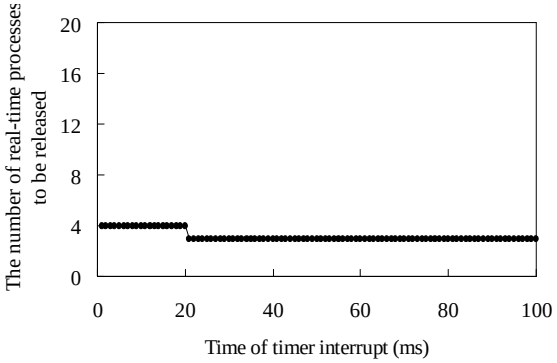

Figure 11. The number of real-time processes to be released at the time of each timer interrupt (*AnT*)

In ART-Linux, when there are many real-time processes with different periods, we speculated that the dispersion of the processing time for the wait is significant. On the other hand, in the proposed control, the value of the processing time for the wait is fixed. Therefore, if there are a large number of real-time processes or many real-time processes with different periods, the effect of the proposed control is high.

When there are a small number of real-time processes, the processing time of the wait for ***AnT*** is longer than that for ART-Linux. The reason may be that the data spatial locality of ***AnT*** is lower than that of ART-Linux, as shown section 4.3. In ART-Linux, the information related to the real-time process is described in only one data structure that is used by a real-time process. Therefore, for the wait, it is unnecessary to refer to the other data structure such as the real-time information table. On the other hand, in the proposed control, the real-time information table is referred to because the release elements are disconnected from the execution management table and the state of the real-time process is changed to a wait state. However, even if the processing time of the wait for ART-Linux is shorter than that for ***AnT***, the value of the difference between ***AnT*** and ART-Linux is small and the value of the processing time for ***AnT*** is fixed. Therefore, we speculated that the proposed control is effective even if there are a small number of real-time processes.

In addition, for periodic execution control, (N - 1) real-time processes require a wait until the N-th real-time process is executed after the timer interrupt. Therefore, the processing time of the wait until the N-th real-time process is executed is the sum total of the processing time for the wait of (N - 1) real-time processes. Hence, the real-time process executed later is significantly affected by the processing time of the wait.

As shown in figure 10, the processing time of the periodic execution control until the 100th real-time process executes is found to decrease by about 50 μs in ***AnT***. This value is about 5% for a period of 1 ms.

4.5. The dispersion for the number of real-time processes to be released

We registered 100 processes with a period of 100 ms, and measured the number of real-time processes to be released and the processing time of the release at the time of each timer interrupt. The process is stopped for x ms when the process is executed until this process requires registration. The value of x is random number ranging from 0 to 99; this value is created by the rand() function of C library. Hence, the timing at which the process requires

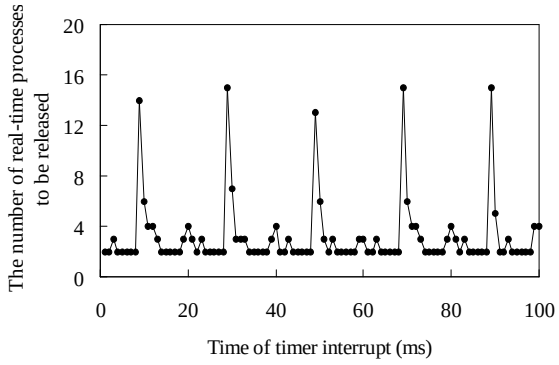

Figure 12. The number of real-time processes to be released at the time of each timer interrupt (ART-Linux)

Table 3. Processing time of release when the number of real-time processes with a period of 100 ms is 100

	AnT	ART-Linux
Maximum	1.50 μs	2.42 μs
Average	1.15 μs	0.66 μs
Minimum	0.95 μs	0.43 μs

registration is irregular. In this measurement, the number of entries in the periodic control table is 100. In this manner, we evaluated effect 4. The number of real-time processes to be released for every timer interrupt for ***AnT*** is shown in figure 11, and that for ART-Linux is shown in figure 12. In addition, the value of the processing time for release is shown in table 3. Moreover, we conducted the measurement 10 times and confirmed that the tendency of the measurement results were identical.

As shown in figure 11, the number of real-time processes to be released for ***AnT*** is 3 or 4. In the proposed control, for registration, when the release element is connected to the periodic control table, the periodic control table is searched, an entry managing few release elements in the periodic control table is found, and the release element of the real-time process is connected to this entry. Therefore, the dispersion for the number of release elements being connected to each entry of the periodic control table is small. In other words, the dispersion for the number of real-time processes in ***AnT*** is small. In addition, owing to the search for the periodic control table to reduce the number of real-time processes to be released at the same time, the results of ***AnT*** show no change when we repeated this measurement.

On the other hand, in figure 12, the number of real-time processes to be released for ART-Linux is large. This is because there is no function to reduce the number of real-time processes to be released at the same time in ART-Linux. Therefore, when the real-time process requires a wait the first time after registration, the number of real-time processes to be released at the same time is decided by the number of real-time processes and the waiting time of each real-time process in the wait queue. In other words, the dispersion for the number of real-time processes at the time of each timer interrupt in ART-Linux is possibly large. Therefore, the results for ART-Linux always showed a change when we repeated the measurements. In other words, if the real-time processes with longer period than the timer interrupt period are registered and the timing of the requirement for registration of the real-

time process is irregular, the dispersion for the number of real-time processes is large in ART-Linux.

As given in table 3, the value of the difference between the maximum value and the average value for ART-Linux is 1.8 μs, but that for ***AnT*** is 0.4 μs, which is small. In other words, when compared to ART-Linux, the dispersion of the processing time for the release in ***AnT*** is small. Therefore, by using the proposed control, the dispersion of the time from the timer interrupt to the execution of a real-time process becomes small.

In addition, we speculated that the timing when the process requires registration is not irregular in practice and many real-time processes requires the registration at the same term. In other words, we believe that the actual dispersion for the number of real-time processes is larger than the result shown in figure 12.

5. Related Work

Many scheduling algorithms have been proposed to realize real-time processing such as periodic processing. Rate monotonic (RM) [9] and earliest deadline first (EDF) [9] have been used as the representative real-time scheduling algorithms on a uniprocessor. As a priority is fixed for RM, the control overhead is small, and it is easy to predict the state transition of the real-time processes. In addition, the dispersion of the interval for the periodic execution of the real-time process with a high priority is small. However, if all real-time processes are completely processed before their deadlines, RM cannot utilize the system completely. EDF is an optimal scheduling algorithm on the uniprocessor because the available system utilization is 100%. In EDF, the dispersion of the interval for the periodic execution of the real-time process with a short period is large; however, there are some cases in which the dispersion of the interval for the periodic execution of the other real-time processes is small. As the priority for EDF is dynamic, the control overhead is large, and it is difficult to predict the state transition of the real-time processes. If the system utilization is high, in RM, the low-priority real-time processes may not be completely processed before the deadline, but the highest-priority real-time process is processed completely. On the other hand, in EDF, there are some cases in which all the real-time processes may not be completely processed before their deadlines. The Least Laxity First (LLF) [11] is also a dynamic priority scheduling algorithm that is used on the uniprocessor.

If the scheduling algorithms are simply adopted on a multiprocessor, then the available system utilization is found to be low. The optimal scheduling algorithms used on multiprocessor are Pfair scheduling [12], T-L plane-based real-time scheduling [13], and EDF with task splitting and k processors in a group (EKG) [14]. In particular, the overhead of Pfair scheduling and T-L plane-based scheduling is large. Therefore, RM zero laxity (RMZL) [15] and RM critical laxity (RMCL) [16] have been proposed to realize scheduling with small control overhead. RMZL and RMCL are based on RM with small control overhead. RMZL and RMCL utilize laxity time that is used by LLF. A scheduling algorithm that combines EDF and LLF is the earliest deadline until zero laxity (EDZL) [17].

These scheduling algorithms can reduce the control overhead by decreasing the number of scheduling events. On the other hand, the sophisticated periodic execution control in this study can reduce the control overhead by reducing the overhead of scheduling event. No studies have been conducted thus far on reducing the control overhead by modifying the implementation method of periodic execution control. A method that effectively uses the context cash for Pfair scheduling [18] was studied for reducing the control overhead by reducing the overhead of scheduling event.

In existing scheduling algorithms, the case in which a real-time process is not completely processed before a deadline is not considered. Currently, to guarantee that all real-time processes are executed completely before the deadline, the worst-case execution time is analyzed and the state where a deadline miss would not occur is built. A method to calculate the worst-case execution time have been previously studied [19][20]. In this paper, for the proposed control, as we believe that a real-time process that is not completely processed before a deadline affects the other real-time processes if such a real-time process is continually executed, a real-time process that is not completely processed before the next release time is terminated.

There are many real-time operating systems based on Linux [21][22][23], such as ART-Linux, which we compared in this paper. Such an operating system is highly versatile because we can use the software library and device driver of Linux. In addition, studies that restrain the influence of interrupt processing on the execution of the real-time processes in Linux-based operating systems [24][25] have been conducted. The scheduler of ART-Linux is based on RM, and the realization of the proposed control is also based on RM. However, we speculated that the proposed control could adopt another scheduling algorithm through expansion of the data structure.

6. Conclusion

We clarified several problems in ART-Linux and proposed a sophisticated periodic execution control to solve these problems. The problems in ART-Linux are as follows: (1) the control overhead for a release is proportional to the number of real-time processes scheduled to be released at the same time, and the dispersion of the processing time for the release is considerable. (2) The control overhead of the connection processing to the wait queue for a wait is proportional to the number of real-time processes, and the dispersion of the processing time for the wait is considerable. (3) A real-time process that is not completely processed before the next release time has an adverse effect on the other real-time processes. (4) The dispersion of the time from the timer interrupt to the execution of a real-time process is significant. (5) The difference in the clock precision between a timer and a processor is not coordinated.

We proposed a method to manage the real-time process scheduled to be released at the time of each timer interrupt as a solution to four of the above-mentioned problems. When the proposed method is used, the control overhead for the release and the wait decrease, and the dispersion of the processing time for them becomes small. In addition, the adverse effect of a real-time process that is not completely processed before the next release time on the other real-time processes is reduced. We have realized the proposed control and explained about the data structure. In addition, we discussed the points to consider while coding. Moreover, we evaluated our suggestion. Though the results of the evaluations, we clarified that the processing time of the release for the proposed control is equal to that for ART-Linux, the processing time of the wait for the proposed control is short and this value is fixed, and the dispersion of the time from the timer interrupt to the execution of a real-time process for the proposed control becomes small. Our future work will involve the development of a method to solve survival problems.

Acknowledgement. This research was partially supported by Grant-in-Aid for Scientific Research 21500055.

References

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

[2] Y. Ishiwata, "SMP kernel based stabilization of ART-Linux and measurement of its real-time processing performance," SI2002 (SICE System Integration), 2002. (in Japanese)

[3] Y. Ishiwata, "Development of ART-Linux on SH-4 processor and its application to quality control," SI2002 (SICE System Integration), 2002. (in Japanese)

[4] Y. Ishiwata, S. Kagami, K. Hishiwaki, T. Matsui, "ART-Linux 2.6 for Single CPU : Design and Implementation," Journal of the RSJ, Vol.26, No.6, pp.546-552, 2008. (in Japanese)

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

[6] Y. Hori, T. Nakajima, T. Katashita, M. Sekiyama, K. TODA, "Approaches to Improving Performance of ART-Linux with Dedicated Hardware," IPSJ Technical Reports, 2004-SLDM-119, Vol.2005, No.27, pp.109-114, 2005. (in Japanese)

[7] ***AnT*** operating system. http://www.swlab.cs.okayama-u.ac.jp/lab/tani/research/AnT/index.html.

[8] K. Okamoto, H. Taniguchi, "Implementation and Evaluation of Fast Inter Server Program Communication for ***AnT*** Operating System," IEICE Transactions on Information and Systems, Vol.J93-D, No.10, pp.1977-1989, 2010. (in Japanese)

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

[10] 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.

[11] J. Y. Leung, "A New Algorithm for Scheduling Periodic, Real-Time Tasks," Algorithmica, Vol.4, pp.209-219 1989.

[12] S. K. Baruah, N. K. Cohen, C. G. Plaxton, D. A. Varvel, "Proportionate progress: A Notion of Fairness in Resource Allocation," Algorithmica, Vol.15, pp.600-625, 1996.

[13] H. Cho, B. Ravindran, E. D. Jensen, "T-L plane-based real-time scheduling for homogeneous multiprocessors," Journal of Parallel and Distributed Computing, Vol.70, Issue 3, pp.225-236, 2010.

[14] B. Andersson, E. Tovar, "Multiprocessor Scheduling with Few Preemptions," Proceedings of 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.322-334, 2006.

[15] A. Takeda, S. Kato, N. Yamasaki, "Real-Time Scheduling based on Rate Monotonic for Multiprocessors," IPSJ Transactions on Advanced Computing Systems, Vol.2, No.1, pp.64-74 (ACS25), 2009. (in Japanese)

[16] S. Kato, N. Yamasaki, "Real-time Scheduling Module for Linux Kernel," IPSJ Transactions on Advanced Computing Systems, Vol.2, No.1 (ACS25), pp.75-86, 2009. (in Japanese)

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

[18] K. Funaoka, S. Kato, N. Yamasaki, "An Effective Use of the Context Cache for Pfair Scheduling," IPSJ Transactions on Advanced Computing Systems, Vol.48, No.SIG 3 (ACS17), pp.1-12, 2007. (in Japanese)

[19] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenstrom, "The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools," ACM Transactions on Embedded Computing Systems, Vol.7, Issue 3, 2008.

[20] K. Yamamoto, Y. Ishikawa, T. Matsui, "Portable Worst-Case Execution Time Analysis Method," IPSJ Transactions on Advanced Computing Systems, Vol.3, No.1 (ACS 29), pp.77-87, 2010. (in Japanese)

[21] TimeSys Corporation: TimeSys Linux, http://www.timesys.com/.

[22] MontaVista Linux. http://www.mvista.com. http://www.montavista.co.jp/.

[23] B. Srinivasan, S. Pather, R. Hill, F. Ansari, D. Niehaus, "A Firm Real-Time System Implementation using Commercial Off-the-Shelf Hardware and Free Software," Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, pp.112-119, 1998.

[24] E. L. Luis, M. Pedro, D. de Niz, "Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware," Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp.14-23, 2006.

[25] M. Dai, Y. Ishikawa, "A Light Interrupt Management for Real-time Linux," IPSJ Transactions on Advanced Computing Systems, Vol.2, No.1, pp.87-95 (ACS25), 2009. (in Japanese)

Authors

Yuuki Furukawa

Received the B.E. degree in 2008 and the M.E. degree in 2010, all from the Okayama University, Okayama, Japan. He has been a doctoral student of Graduate School of Natural Science and Technology at Okayama University since 2010. His research interests include operating system and real-time processing. He is a member of the Information Processing Society of Japan (IPSJ).

Toshihiro Yamauchi

Received the B.E. degree in 1998, the M.E. degree in 2000, and the Ph.D. degree in computer science in 2002, all from the Kyushu University, Fukuoka, Japan. In 2001 he was a Research Fellow of the Japan Society for the Promotion of Science. In 2002 he became a Research Associate in Faculty of Information Science and Electrical Engineering at Kyushu University. He has been an associate professor of Graduate School of Natural Science and Technology at Okayama University since 2005. His research interests include operating systems and computer security. He is a member of the Information Processing Society of Japan (IPSJ), the institute of Electronics, Information and Communication Engineers (IEICE), USENIX, and ACM.

Hideo Taniguchi

Received the B.E. degree in 1978, the M.E. degree in 1980, and the Ph.D. degree in 1991, all from the Kyushu University, Fukuoka, Japan. In 1980, he joined NTT Electrical Communication Laboratories. In 1988, he moved Research and Development headquarters, NTT DATA Communications Systems Corporation. He had been an Associate Professor of Computer Science at Kyushu University since 1993. He has been a Professor of Faculty of Engineering at Okayama University since 2003. His research interests include operating system, real-time processing and distributed processing. He is the author of "Operating Systems" (Shoko Publishing Co. Ltd.) etc. He is a member of the Information Processing Society of Japan (IPSJ), the institute of Electronics, Information and Communication Engineers (IEICE), and ACM.