GUARANTEE OF SERVICE PROCESSING TIME OF PROCESS GROUP FOR MULTIMEDIA APPLICATION

Toshihiro TABATA, Yoshinari NOMURA and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering Kyushu University,
Fukuoka 812-8581, Japan
tabata@swlab.csce.kyushu-u.ac.jp, {nom, tani}@csce.kyushu-u.ac.jp

ABSTRACT

Computer performance has improved year by year. As a result, multimedia applications can be executed on a computer simultaneously. However multimedia applications sometimes can not provide a good service because multimedia applications compete with themselves to get computer resources. For this reason, computer resources need to be reserved for multimedia applications to provide a good service. Besides, there are a lot of service applications that consist of multi process. Thus multi process need to be a unit of resource reservation to provide a service of appropriate quality. We call multi process a process group. In this paper, we propose *execution* that is a unit of processor time assignment. In addition we propose a mechanism for guaranteeing processing time of process groups by *executions*.

1. INTRODUCTION

Multimedia applications that play movies and sounds require much computer resources. Especially processor time assignment is important for multimedia applications to provide a good service because multimedia applications need a lot of processing time. Thus multimedia applications need to be guaranteed it's processing time. However it is difficult to guarantee processing time of multimedia applications by conventional time-sharing scheduling. As a result, multimedia applications compete with themselves to get computer resources when they are executed on the same computer simultaneously.

Multi process need to be a unit of guarantee of computer resources because most of applications consist of multi process. However it is difficult to guarantee computer resources of applications because multi process should be a unit of resource reservation. Also, number of processes which compose service applications, is not constant. For this reason, it is difficult to specify total amount of computer resources that are guaranteed to a service application when one process is a unit of resource reservation.

In this paper, we propose *execution* that is a unit of processor time assignment. In addition we propose a mechanism for guaranteeing processing time of a program by an *execution*. *Execution* enables an operating system to regulate execution speed and to guarantee processing time of a program. We call degree of assigning processor time "degree of processor assignment". *Execution* has "degree of processor assignment". Two types of degree of processor assignment are realized in *execution*. To represent process groups by hierarchical *execution* enables an operating system to guarantee service-processing time of process groups.

This paper is organized as follows. Details of *execution* are described in the next section. In Section 3, hierarchical *execution* is presented. In Section 4, basic evaluation is discussed. In Section 5, related works are described. Finally some conclusion and remarks are given in Section 6.

2. EXECUTION

2.1. Overview

A process is a unit of program execution in conventional operating systems. A process has degree of processor assignment. For example, a priority is associated with each process in UNIX and the processor is allocated to the process with the highest priority. Hence a process may be executed soon after the process was created.

We separated degree of processor assignment from a process. Degree of processor assignment was named *execution*. The remains of a process are named a "*process*". A *process* does not have degree of processor assignment. An *execution* has degree of processor assignment.

Scheduling queues appear in Figure 1. Before we introduce *executions*, processes are kept on linked list that corresponds to a priority of a process. After we

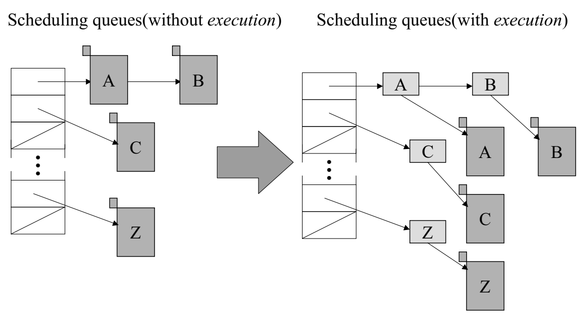
Scheduling queues(without *execution*)
Scheduling queues(with *execution*)
A
B
C
Z
A
B
C
A
B
C
Z
Z

Figure 1: Change of scheduling queues.

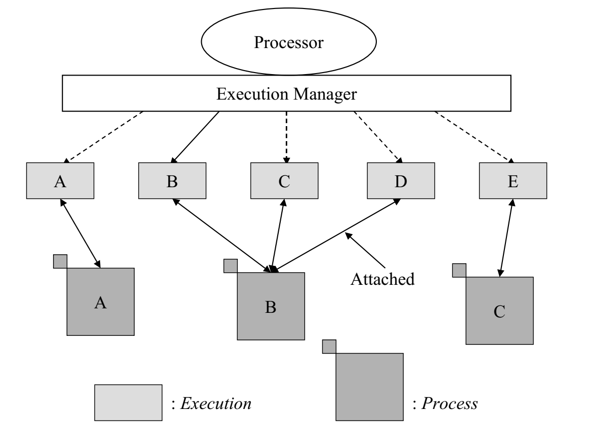

Figure 2: Relation between *process* and *execution*.

introduce *executions*, *executions* are kept on linked list that corresponds to a priority of *executions*. *Processes* are linked to *executions*.

Execution enables an operating system to reserve *processes* without allocation of processor. A *process* can be executed by attaching an *execution*. Thus creating processes in an idle processor time can speed up process creation.

Figure 2 shows the relation between *executions* and *processes*. Multi *execution* can be attached to a *process*. The example is process B in Figure 2. Execution manager points to an *execution* with the highest degree of processor assignment. All *processes* need to be attached to *executions* to be assigned processor time.

Execution manager selects a *process* in scheduling queues. The *process* is executable and is attached to *execution* with a highest priority. The amount of processor time that is assigned to a *process* is in proportion to the total amount of degree of processor assignment of *executions* that is attached to the *process*.

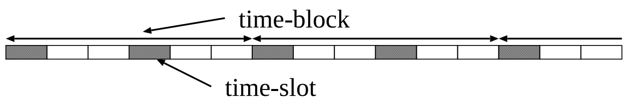

Figure 3: Relation between time-slots and a time-block.

2.2. Types of Execution

There are two types of *execution*. Those are *execution* with performance and *execution* with priority.

2.2.1. Execution with Performance

Execution with performance has degree of processor assignment that indicates a ratio (%) to processor bare performance. The processor bare performance can be defined as 100%. When a *process* is attached to an *execution* with n%, the assigned processor time is n% of the processor bare performance.

We describe the mechanism of assigning processor time as follows.

綉 ime-slot and Time-block

An operating system can regulate processor assignment by controlling a state of a *process*. We named a unit of processor assignment “time-slot”. We named a chunk of time-slots “time-block”. Figure 3 shows the relation between time-slots and a time-block. An operating system assigns processor to an *execution* that is assigned to current time-slot. If current time-slot is not assigned to any *process*, an operating system executes a *process* with *execution* that has a highest priority.

An *execution* with n% degree of processor assignment is assigned n% of time-slots in a time-block. For example, a time-block consists of six time-slots in Figure 3. Also, two time-slots are assigned to an *execution* in a time-block. As a result, the *execution* is assigned 33% of processor bare performance.

絎 ethod of Assigning Time-slots

Degree of regulating program execution speed can be evaluated by the processing time of program. However the processing time of program is not enough to evaluate degree of regulating program execution speed because the processing time is measured between start and finish of processing. What is important is to evaluate uniformity of processing between start and finish of processing. If behavior of the processing is not smooth, the *process* can not provide a service of a good quality. In order to execute a process smoothly, it is necessary to execute regulated the processing in a constant interval.

We proposed a new cyclic like assignment method[1]. When n time-slots are assigned, *(i-th assigned time-slot*

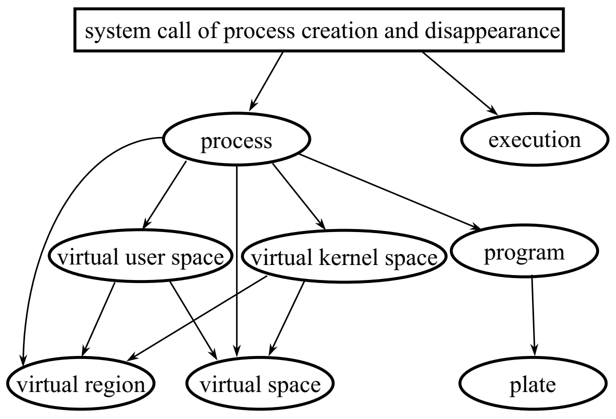

Figure 4: Relation among resources on process creation and disappearance.

*position) = (total number of time-slots in a time-block) * (i–1) / n.* The method can assign time-slots at a constant interval. Therefore uniformity of processing is best in our proposed method.

Many multimedia applications are interactive. For example, a multimedia application that plays a movie shows the motion of picture to a user. The application has to be assigned processor time in a constant interval to provide a good service. Otherwise, the motion of picture does not move smoothly. Our mechanism would be effective for multimedia applications.

2.2.2. Execution with Priority

Execution with priority has degree of performance assignment that indicates a priority. *Execution* with priority that has a highest priority is assigned processor. However *execution* with performance is taken precedence over *execution* with priority because assigned processor time of *execution* with performance have to be guaranteed.

2.3. Implementation

We implemented *execution* in ***Tender*** operating system[2] that we have been developing. In this subsection, we describe overview of ***Tender*** operating system.

In ***Tender***, objects to be controlled and managed by operating system are called "resources". We realized high modularity of the resources. As a result, each resource can exist irrespective of existence of other resources. ***Tender*** manages resources by resource identifiers and resource names. Besides, resources are controlled by unified interface.

The relation among the resources in the processing of process creation and deletion is shown in Figure 4. Arrows show the relation among the resources. A process of a conventional operating system was separated to 8 resources. Plate corresponds to file.

The separation and individualization of the process resources clarified the relation between *process* and elements of process. Furthermore, it realized that the elements of a process could exist irrespective of existence of *process*. For example, all of memory space is released when a process disappears in UNIX. Therefore, the memory space is not recycled. In ***Tender***, the memory space that is designated can be preserved when the process disappears. If the memory space is recycled on process creation, the processing of process creation is sped up,

2.4. Interface

Table 1 shows interface of *execution*. 8 system-calls are implemented on ***Tender***.

2.5. Merit of Execution

Merits of *execution* are described as follows[3].

(1) Fast process creation and deletion.

(2) Suspension and resume a processing by attachment and detachment of an *execution*.

(3) Process restarting by initializing only data segment.

In addition, merits of attachment of multi *execution* to a *process* are described as follows.

(1) User level scheduling by attachment of multi *execution*.

(2) Increase of assigned processing time by attachment of multi *execution* with priority.

(3) Guarantee in processing time by attachment of *executions* with performance and *executions* with priority.

3. HIERARCHICAL EXECUTION

We expressed *executions* with tree structure to guarantee the processing time of a process group. We describe hierarchical *execution* in this section.

3.1. Requirement

Requirements for hierarchical *execution* are described as follows.

(1) A process group is expressed by one *execution*.

Table 1: Interface of *execution*.

No.	Form	Contents of operation
1	creat_execution(mips)	*Execution* is created and execution identifier is returned. When mips is from 1 to 100, the *process* runs with the mips% of bare processor performance. When mips is from -255 to 0, mips means priority.
2	delete_execution(execid)	*Execution*(execid) is deleted.
3	attach_execution(execid, pid)	*Execution*(execid) is attached to *process*(pid).
4	detach_execution(execid, pid)	*Execution*(execid) is detached from *process*(pid).
5	wait_execution(pid, chan)	Assignment of processor time to *process*(pid) is stopped. The *process* state changes to WAIT state. If pid equals 0, pid means current running *process*. Chan means identifier of WAIT.
6	wakeup_execution(pid, chan)	Assignment of processor time to *process*(pid) is restarted. The *process* state changes to READY state. If pid equals 0, the *processes* with chan are restarted. If chan equals 0, chan means all identifier of WAIT.
7	dispatch(pid)	*Process*(pid) is run.
8	control_execution(execid, mips)	Degree of processor assignment of *execution* (execid) is changed to mips.

(2) Each *process* in a process group has degree of processor assignment.

(3) Degree of processor assignment of a process group is indicated same as a *process*.

3.2. Relation between Execution and Process Group

Process group has degree of processor assignment, indicated by *executions* because process group is to be controlled like a *process*. Figure 5 shows relations between *process* and *execution*. One *process* can be attached to more than one *execution* (1). A service application consists of three *processes* (2). The service application can be attached to more than one *execution* (3).

Figure 6 shows relation between each *process* and *execution*. Each *process* of process group is attached to more than one *execution*. Thus, each *process* has degree of processor assignment.

3.3. Mechanism of Hierarchical Execution

In this subsection, basic mechanism of hierarchical *execution* is described. Also, modified interface and scheduling mechanism are described.

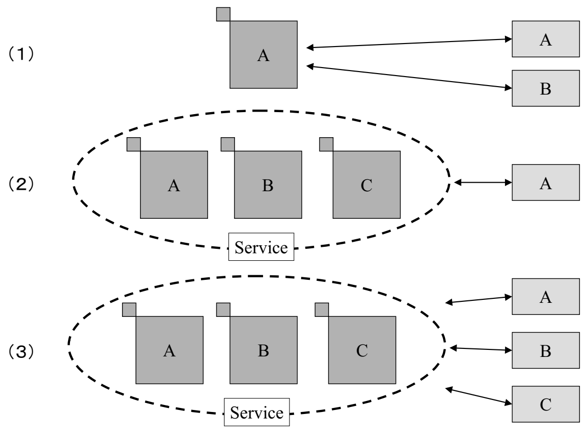

Figure 5: Relation between *execution* and process group.

3.3.1. Basic Mechanism

Execution manager requires information of process groups and *processes* for process scheduling. For this reason, process groups are represented by *executions*. As a result, the execution manager can have the information. It prevents the overhead of scheduling from increasing.

Structure of a process group is represented in tree structure of *executions* because relation between a pro-

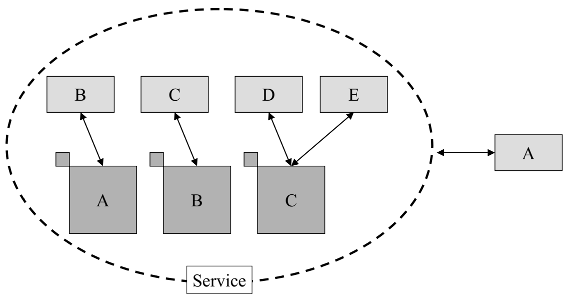

Figure 6: Relation between *execution* and each *process* of process group.

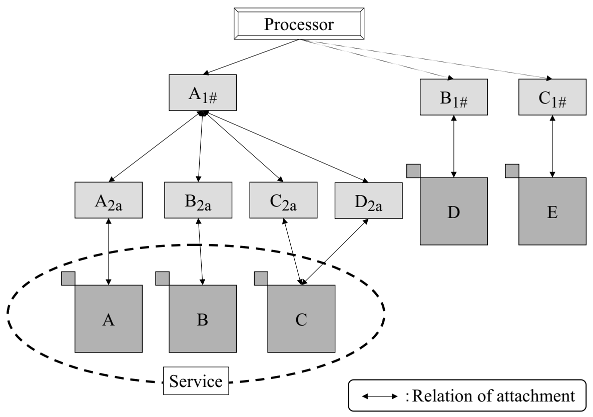

Figure 7: Representation of process group by *executions.*

cess group and *processes* is represented in parent and child. Figure 7 shows relations between a process group and *executions.* Node of *execution* tree is called "directory *execution*". Directory *execution* represents a process group. Leaf is called "leaf *execution*". Leaf *execution* is attached to a *process.*

The total amount of assigned processor time of leaf *executions* equals assigned processor time of parent directory *execution.* Degree of processor assignment of leaf *executions* indicates a priority or a ratio (%) to assigned processor time of parent directory *execution.* In the leaf *execution*, the ratio is indicated where the parent directory *execution* is defined as 100%.

Depth of *execution* tree is more than one. As a result, it is possible to create a process group into other process group. Figure 8 shows *execution* tree when the depth of *execution* tree is n. In the case, a directory *execution* is attached to other directory *execution.*

Figure 9 shows a case that more than one *execution* is assigned to a process group. When second execution ($B_{1\#}$) is attached to a process group, leaf executions (D_{2b}, E_{2b}, F_{2b}) have to be created and attached to each

Figure 8: Hierarchical *execution* tree(n depth).

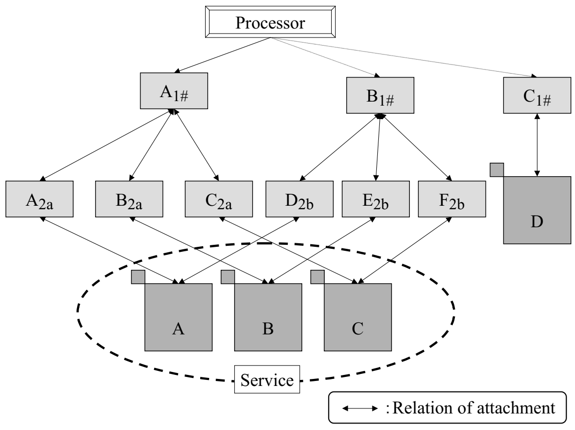

Figure 9: Two process group executions.

process (A, B, C) in the process group. As a result, each process of the process group is attached to two leaf *executions.*

3.3.2. Modified Interface

Two interfaces are modified. The modified interfaces are relation to operations of attachment and detachment. Table 2 shows the modified interfaces.

3.3.3. Process Scheduling Mechanism

Figure 10 shows process-scheduling mechanism. Scheduler is executed by timer interrupt. First, the scheduler checks next time-slot. If the time-slot is not attached to any *executions*, the scheduler chooses an *execution* with a highest priority to run. Otherwise the scheduler checks whether the *execution* is directory *execution.* Next, if the *execution* is directory *execution*, the scheduler checks directory *execution* recursively. Oth-

Table 2: Additional *execution* interface for *execution* tree.

No.	Form	Contents of operation
3'	attach_execution(execid, rid)	Rid means process identifier(pid) or execution identifier(execid2). (1) If rid means pid, *execution* (execid) is attached to *process* (rid). (2) If rid means execid2, *execution* (execid) is attached to *execution* (execid2). *Execution* (execid2) is parent of *execution* (execid).
4'	detach_execution(execid, rid)	Rid means process identifier(pid) or execution identifier(execid2). (1) If rid means pid, *execution* (execid) is detached from *process* (rid). (2) If rid means execid2, *execution* (execid) is detached from *execution* (execid2). *Execution* (execid2) belong to first class.

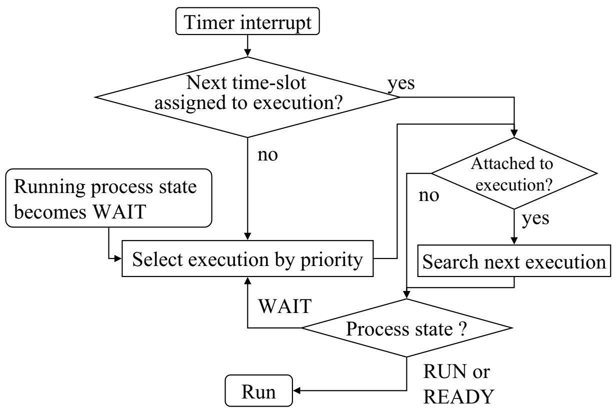

Figure 10: Flow of process scheduling.

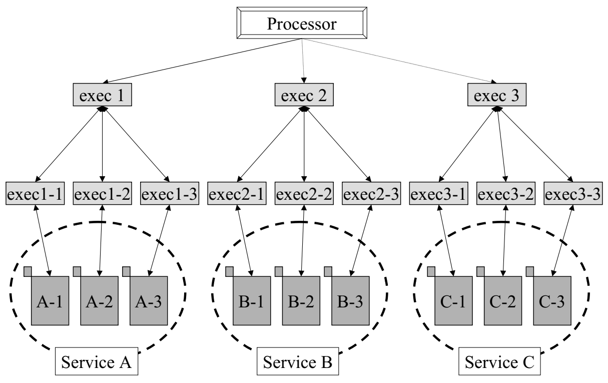

Figure 11: Test case for evaluation.

erwise the scheduler checks whether the state of the *process* that is attached to the *execution*, is RUN or READY. If the *process* is RUN or READY, a processor time is assigned to the *process*.

3.4. Merit of Hierarchical Execution

Merits of hierarchical *execution* are described as follows.

(1) Attachment of *execution* with performance enables a service application to regulate its execution performance.

(2) Attachment of *execution* with performance and *execution* with priority enables a service application to guarantee its processing time.

(3) Hierarchical *execution* tree can represent structure of service applications.

4. EVALUATION

We measured processing time of process groups. Figure 11 shows test case for evaluation. There are three pro-

Table 3: Degree of processor assignment of service A and C.

service A		service C	
exec1	30%	exec3	6
exec1-1	20%	exec3-1	50%
exec1-2	50%	exec3-2	6
exec1-3	6	exec3-3	6

Table 4: Degree of processor assignment of service B.

case	exec2	exec2-1	exec2-2	exec2-2
1	40%	30%	40%	6
2	40%	30%	40%	–
3	40%	30%	–	6
4	40%	–	–	6
5	40%	–	–	–

cess groups. Each process group has three *processes*. The processing of the *processes* is using processor only. Table 3 shows that degree of processor assignment of service A and C. Table 4 shows that degree of processor assignment of service B in each case. We evaluated the five test cases that are shows Table 4. Degree of processor assignment of *executions* that consist of service B changed in each case. Degree of processor assignment of *executions* that consist of service A and C are constant in each case. Results of measurement appear in Figure 12 and 13.

We found followings from the result of Figure 12:

(1) Ratio to processing time of process group A was constant. *Execution* with performance can regulate performance of a process group.

(2) Ratio to processing time of process group B was 40% when exec2-3 existed. On the other hand, the ratio to processing time of process group B was less than 40% when exec2-3 did not exist. Since *execution* with performance did not require performance as much as possible, process group B abandoned remains of processing time. In case 2, the remains of processing time is 12% (= 40% * (100% - (30% + 40%))).

(3) Processing time that is not assigned to process group A and B is assigned to process group C.

(4) Ratio to processing time of each process group is in accordance with degree of processor assignment. Also, ratio to processing time in each *process* on each process group is in accordance with degree of processor assignment.

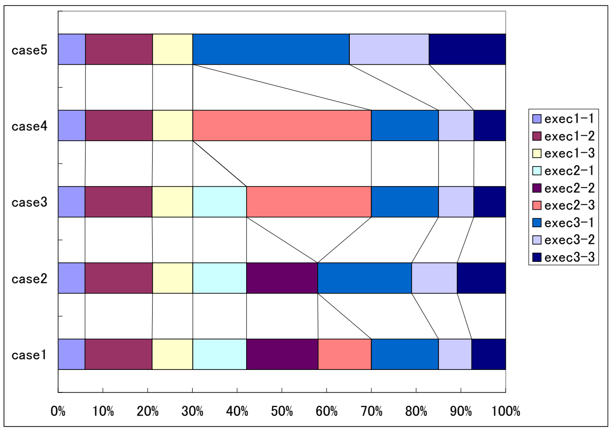

Figure 12: Result of measurement (one execution is attached to service A).

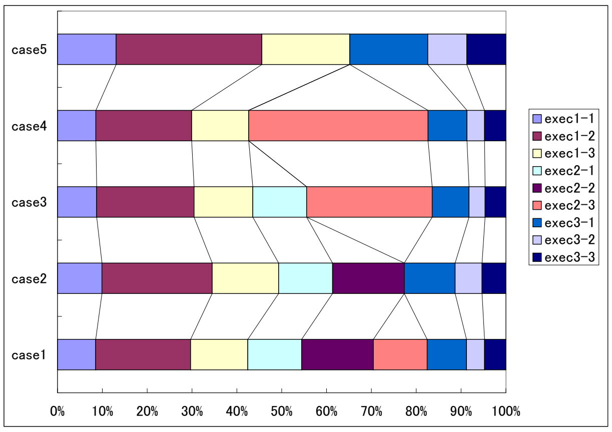

Figure 13: Result of measurement when two executions are attached to service A.

Next, we attached another *execution* with priority to process group A. The priority is 6. Also, we attached three leaf *executions* to *processes* in process group A. Degree of processor assignment of the leaf *executions* is the same as exec2-1, exec2-2 and exec2-3. The results of evaluation appear in Figure 13. The following results were obtained from Figure 13:

Processing time of process group A is approximately equal to sum of 30% to processor time and processing time of process group C. That is, processing time of process group C indicates processing time of *execution* with priority. Thus processing time of process group A can be ensured at least 30%.

5. RELATED WORK

Several studies have been made on process scheduling mechanism. A feedback-driven approach is used for real-time application[4]. Also, some relevant works are based on period and proportion [5][6]. Proportional scheduling[7] presents relative execution rates of computation as degree of processor assignment. Our proposed mechanism presents a rate to bare processor performance as degree of processor assignment.

Hierarchical scheduler[8] has some scheduling mechanism. And a scheduling mechanism can be applied to a process. Our proposed mechanism has some scheduling mechanism. Besides, it can apply some scheduling mechanisms to one process at the same time.

6. CONCLUSION

In this paper, we propose a guarantee mechanism of service processing time of process groups by *execution*. Processor time assignment unit is called "*execution*". *Execution* has two types of degree of processor assignment that are "*execution* with performance" and "*execution* with priority". Attachment of *executions* with performance and *executions* with priority enables an operating system to guarantee processing time of *processes*.

Structure of process group is represented in tree structure of *execution*. Depth of *execution* tree is more than one. We described three merits of hierarchical *execution* as follows. Attachment of *execution* with performance enables service applications to regulate its execution performance. Attachment of *execution* with performance and *execution* with priority enables service applications to guarantee its processing time. Hierarchical *execution* tree can represent structure of service applications.

We measured processing time of process groups. The following results were obtained from measurements: *Execution* with performance can regulate performance of process groups. Ratio to processing time of each process is in accordance with degree of processor assignment. Processing time of process group can be ensured by *execution* with performance.

As a future work, we will evaluate the mechanism on a service application.

7. REFERENCES

[1] TABATA, T., TANIGUCHI, H. and USHIJIMA, K.: Method of Improving Uniformity of Processing on Program Speed Control Mechanism, Proceedings of 59th National Conventions of IPSJ, No. 1, pp.37–38 (1999(in japanese)).

[2] TABATA, T. and TANIGUCHI, H.: Guarantee of Service Processing Time by Execution on Tender Operating System,, *Transactions of IPSJ*, Vol.41, No.6, pp.1745–1754 (2000(in japanese)).

[3] TABATA, T. and TANIGUCHI, H.: Implementation and Evaluation of Speed Control Mechanism of Program Execution on Resource Execution on Tender, *Transactions of IPSJ*, Vol.40, No.6, pp.2523–2533 (1999(in japanese)).

[4] Steere, D. C., Goel, A., Gruenberg, J. and McNamee, D.: A Feedback-driven Proportion Allocator for Real-Rate Scheduling, OSDI'99: USENIX Association 3rd Symposium, pp.145–157 (1999).

[5] Nieh, J. and Lam, M. S.: The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications, SOSP'97: 16th ACM Symposium on Operating Systems Principles, pp.184–197 (1997).

[6] Jones, M. B., Rosu, D. and Rosu, M.-C.: CPU Reservations and Time Constraint: Efficient, Predictable Scheduling of Independent Activities, SOSP'97: 16th ACM Symposium on Operating Systems Principles, pp.198–211 (1997).

[7] Waldspurger, C. A. and Weihl, W. E.: Lottery Scheduling: Flexible Proportional-Share Resource Management, OSDI'94: USENIX Association 1st Symposium, pp.1–11 (1994).

[8] Goyal, P., Guo, X. and Vin, H. M.: A Hierarchical CPU Scheduler for Multimedia Operating Systems, OSDI'96: USENIX Association Second Symposium, pp.107–121 (1996).