Noname manuscript No.
(will be inserted by the editor)

A Mechanism for Achieving a Bound on Execution Performance of Process Group to Limit CPU Abuse

Toshihiro Yamauchi · Takayuki Hara · Hideo Taniguchi

Received: date / Accepted: date

Abstract The secure OS has been the focus of several studies. However, CPU resources, which are important resources for executing a program, are not the object of access control in secure OS. For preventing the abuse of CPU resources, we had earlier proposed a new type of execution resource that controls the maximum CPU usage [8]. The previously proposed mechanism can control only one process at a time. Because most services involve multiple processes, the mechanism should control all the processes in each service. In this paper, we propose an improved mechanism that helps to achieve a bound on the execution performance of a process group in order to limit unnecessary processor usage. We report the results of an evaluation of our proposed mechanism.

Keywords Process scheduling, operating system, anti-DoS technique, execution resource, security

1 Introduction

The number of computers connected to a network has increased with the widespread use of the Internet. In addition, the number of reports of software vulnerabilities has been increasing every year. This increase can be attributed to the widespread use of automated attack tools and the increasing number of attacks against systems connected to the Internet [1]. For example, viruses and worms spread rapidly through the Internet, causing serious damage to many computers. In addition, Denial of Service (DoS) attacks are one of the serious problems encountered by computer systems. Therefore, various defense

Toshihiro Yamauchi · Takayuki Hara · Hideo Taniguchi
Graduate School of Natural Science and Technology, Okayama University,
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
Tel.: +81-86-251-8188
Fax: +81-86-251-8256
E-mail: {yamauchi, tani}@cs.okayama-u.ac.jp

mechanisms against such attacks have been studied extensively, and these studies have gained considerable attention.

Various defense mechanisms include firewalls, an Intrusion Detection System (IDS) [2–4], buffer overflow protection and access control mechanisms such as Mandatory Access Control (MAC) and Role-Based Access Control (RBAC) [5], and the secure OS. A firewall can block packets from incoming connections, but it cannot evaluate the content of "legitimate" packets. Thus, a firewall cannot prevent attacks from passing through legitimate ports. Besides, IDSs observe program execution and detect malicious behavior of the program. The incidence of false negatives and false positives on IDS is not zero. Therefore, the combination of a firewall and IDS is not sufficient to prevent attackers from attacking computers.

The secure OS [6] has been the focus of several studies. In particular, Security-Enhanced Linux (SELinux) has attracted considerable attention. Even if the authority is taken, the secure OS ensures minimal influence. However, the CPU resource, which is an important resource for executing a program, is not the object of the access control in secure OS. As a result, such OSes cannot control the CPU usage ratio. For example, a secure OS cannot prevent attackers from carrying out DoS attacks, which affect the CPU resources. In general, the OSes can only limit the maximum CPU time for each process and not the proportion of CPU time allocated to the processes.

Most OSes cannot control the usage ratio of CPU resources for users or programs appropriately. If the ratio is controlled appropriately and easily, the impact of DoS attacks can be mitigated. For example, if an administrator can reserve an amount of CPU resource for administration, he/she can perform maintenance activities on the computer, where it is being attacked. In addition, an administrator can limit the usage ratio of CPU resource for each user and stop the allocation of CPU to malicious users.

In an earlier study, we proposed the execution resource as a unit of program execution, which has a degree of CPU usage[7]. In addition, we proposed a new type of execution resource that controls the maximum CPU usage so that the abuse of CPU resources can be prevented [8]. In order to prevent this abuse, we propose an execution resource that can limit the upper bound of CPU usage. The execution resource can specify the amount of CPU usage for each process. In other words, execution can limit a proper amount of CPU usage within a service that consists of two or more processes. That is, this execution resource can be applied to mitigate DoS attacks.

The previously proposed mechanism can control only one process at a time. Because, most service involve multiple processes, the mechanism should control all the processes involved in each service. In this paper, we propose an improved mechanism for achieving a bound on the execution performance of a process group in order to limit unnecessary processor use. The main idea of this proposed mechanism is that the proposed mechanism limits the assigned CPU time of attack processes in order to guarantee the execution of important services. Here, the proposed mechanism is based on a previously proposed mechanism. The previous mechanism could be used only to control a process

because the execution resource with an upper bound was a leaf execution. The proposed mechanism introduces directory execution as an execution resource with an upper bound for the nodes of the execution tree. Because the proposed mechanism can control the upper bound of a directory execution resource attached to a process group, the abuse of CPU time of malicious processes in the process group can be efficiently restricted. We also describe the implementation of the proposed mechanism on The ENduring operating system for Distributed EnviRonment (***Tender*** operating system) [9]. In addition, this paper presents the results for a case involving an attack and the evaluation results obtained using the Apache web server.

2 Execution Resource

In this section, we explain the concept of execution resource on the basis of the presentation and previously proposed concept of execution resource in other papers[7, 8].

2.1 Overview

A process may be described as a unit of program execution in an existing OS, and it has a degree of CPU usage. For instance, a priority is associated with each process in UNIX. We have separated the degree of CPU usage from a process. The degree of CPU usage is termed as the execution resource. Therefore, only the execution resource involves the degree of CPU usage, and a process does not have a degree of CPU usage. Scheduling queues are shown in Fig. 1. Prior to the introduction of the execution resources, processes are listed on a linked list on the basis of their priority. After the introduction, these execution resources are maintained on the linked list on the basis of their priority. Processes are then linked to executions. A process can be executed by linking it to an execution.

Fig. 2 shows the relation between executions and processes. The execution manager points to an execution with the highest degree of CPU usage. All processes need to be linked to executions in order to be assigned a CPU time. The execution manager selects a process from the scheduling queues. When the state of a process is READY, it is linked to the execution with the highest priority. The amount of CPU time that is assigned to a process is proportional to the total amount of CPU usage time required for the executions linked to the process.

2.2 Types of Execution Resources

There are two types of execution resources. One is execution with performance and the other is execution with priority.

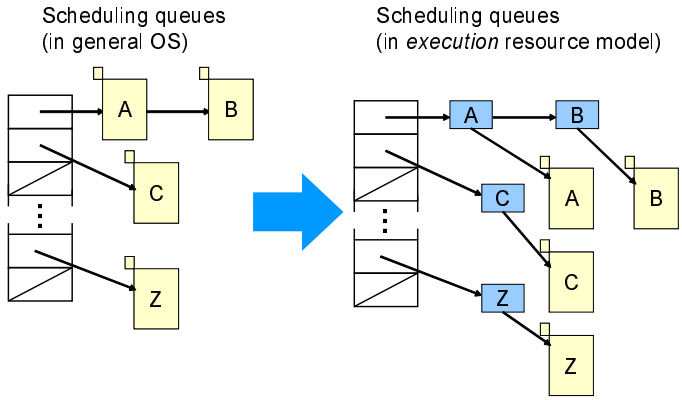


Fig. 1 Change of scheduling queues

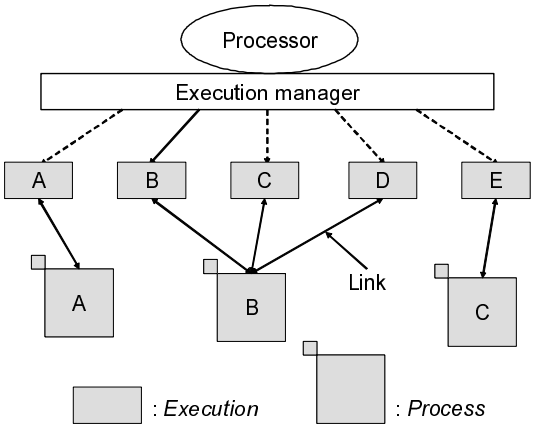


Fig. 2 Relation between executions and a process

2.2.1 Execution with performance

Execution with performance includes the degree of CPU usage that indicates the proportion of bare processor performance. The bare processor performance can be set as 100%. When a process is linked to an execution with n% performance, the assigned CPU time is n% of the bare processor performance. We termed a unit of CPU usage as a "time slot," and we termed a group of time slots as a "time block." Fig. 3 shows the relation between time slots and a time block. For an execution in which the degree of CPU usage is n%, n% of the time slots are assigned in a time block.

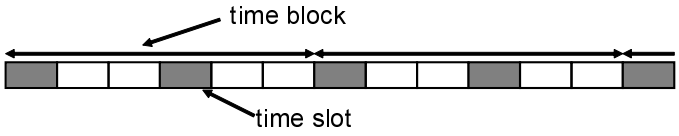


Fig. 3 Time slots and a time block

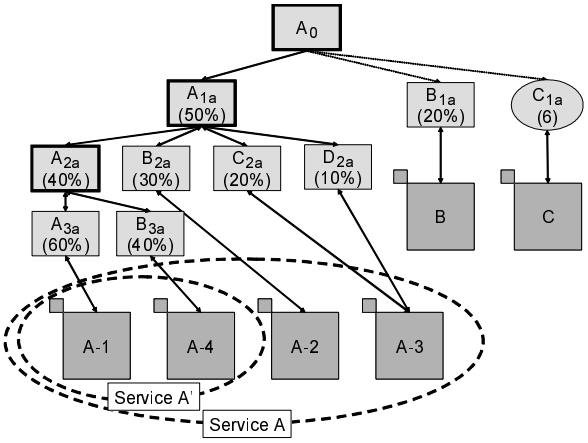


Fig. 4 Relationships between executions and a process group

2.2.2 Execution with priority

Execution with priority includes the degree of performance that indicates the priority. The execution manager assigns the execution with priority that has the highest priority to the processor. However, execution with performance takes precedence over execution with priority because the former is guaranteed an assigned CPU time.

2.3 Hierarchical Execution Tree

The structure of a process group is represented as a tree structure of executions because the relation between a process group and its processes is represented as a parent and a child. Fig. 4 shows the relationships between a process group and executions. The root of an execution tree is called "root execution" and it represents the bare processor performance. The node of an execution tree is called "directory execution" and it represents the degree of CPU usage for a process group. A leaf is called "leaf execution"; every leaf execution is linked to a process. The execution resource (C_{1a}) is an execution with priority. The priority of it is 6. The other execution resources shown in Fig. 4 are execution with performance.

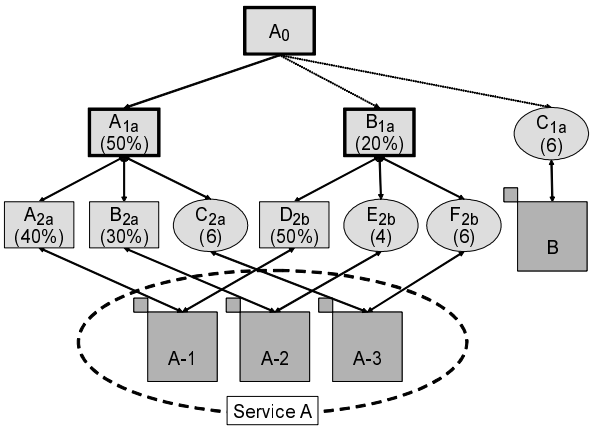


Fig. 5 Two process group executions

The total assigned CPU time for leaf executions equals the assigned CPU time for the parent directory execution. The degree of CPU usage for leaf executions indicates the priority or the ratio (%) to the assigned CPU time of the parent directory execution. In the leaf execution, the ratio corresponds to the point where the parent directory execution is set as 100%. The depth of an execution tree is greater than one. As a result, it is possible to create a process group within another process group.

Fig. 5 shows a case where more than one execution is linked to a process group. When the second execution (B_{1a}) is linked to a process group, leaf executions (D_{2b}, E_{2b}, and F_{2b}) have to be created and linked to each process (A, B, and C, respectively) in the process group. As a result, each process within the process group is linked to two leaf executions.

2.4 Operation Interface of Execution Resource

We designed eight operation interfaces for the execution resource to construct an execution tree and control a program execution. Table 1 shows the interfaces.

3 Execution Resource with an Upper Bound

3.1 Rate-Limiting Mechanism Based on the Use of Execution Resources

In the existing OSes, whose operation is based on the time-sharing technique, the CPU time used by a process according to its priority does not have an upper bound. Therefore, the allocation of CPU time to other services is affected when two or more programs that demand infinite CPU time run simultaneously. In this case, service performance deteriorates significantly.

Table 1 Operation interfaces of the execution resource

Form	Contents of operation
creat_execution (mips)	Create the execution specified by *mips* and return the execution identifier *execid*. When *mips* is between 1 and 100 it signifies the performance regulation execution degree (as a percentage with the performance of the processor itself taken to be 100 percent), when it is 0 or negative is signifies the priority of the execution degree (the absolute value is the process priority).
delete_execution (execid)	Delete the execution *execid*.
attach_execution (execid, rid)	If *rid* means pid, this interface associates the execution *execid* and the process *rid*. If *rid* means execid, this interface associates the execution *execid* and the execution *rid*.
detach_execution (execid, rid)	If *rid* means pid, this interface remove the association between execution *execid* and process *rid*. If *rid* means execid, this interface remove the association between execution *execid* and execution *rid*.
wait_execution (pid, chan)	Forbid the assignment of processor [time] to process *pid* and its associated execution[s]; this puts the process in the WAIT state.
wakeup_execution (pid, chan)	Make it possible to assign CPU time to the process *pid* and its associated executions; this puts the process in the READY state.
dispatch(pid)	Run process *pid*.
control_execution (execid, mips)	Change the execution degree of *execid* to *mips*. *mips* is interpreted as in creat_execution.

To prevent the abuse of the CPU resources, we proposed an execution resource that helps to achieve an upper bound for the CPU usage ratio[8]. The main idea of this proposed mechanism is that the proposed mechanism limits the assigned CPU time of attack processes in order to guarantee the execution of important services. Important services and legitimate services are attached to execution resource without an upper bound. Because the assignment of CPU time for these services is not limited by the proposed mechanism, these services can run according to their priority. On the other hand, suspicious services are attached to execution resource with an upper bound. If the suspicious services attack a computer, the proposed mechanism can limit the upper bound of the assigned CPU time of the services. Thus, the damage of such attacks can be mitigated. In addition, the upper bound of suspicious services can be changed to lower value by an administrator after the attacks are detected.

Next, the process flow of limiting assigned CPU time is explained. In this execution resource with an upper bound, the CPU time is allocated according to the priority until the usage reaches a specified ratio in a time slice. The time slice is the period of time for which a process is allowed to run. When it reaches the specified ratio, the state of the currently running process is changed to a WAIT state until the current time slice expires. Even if a process that is linked to an execution for which the CPU usage is limited by an upper bound suffers a malicious attack, the execution system can prevent the program from using excessive CPU time. Moreover, the execution resource can be grouped with a user or a service. Therefore, the CPU usage ratio of a user or a service can be specified. As a result, the impact of a DoS attack can be controlled within the

process group even if a new child execution is created because the execution belongs to the same group.

As described in a previous paper [7], the previously proposed mechaism can guarantee that the important processes will be effectively carried out by using the execution resources with high performance.

3.2 Execution Resource with an Upper Bound for Process Group

In the previous mechanism, the execution resource with an upper bound was a leaf execution. The previous mechanism could be used only to control a process. Therefore, in order to control the upper bound of a process group, a user must change the upper bound of each process. If a process group consists of 100 processes, the operation should be performed 100 times. It increases the load on the user side. In addition, the previous proposed mechanism cannot limit sufficiently the upper bound of a process group because it cannot limit the upper bound of the process group as a whole. Therefore, we introduce directory execution as an execution resource with an upper bound. Directory execution represents the degree of CPU usage for a process group. Thus, the proposed mechanism can control the upper bound of a process group as a whole by introduction of upper bound for directory execution. As a result, directory execution with an upper bound can efficiently limit the influence of the attack processes.

In order to do so, the process scheduler was changed for the control of an execution resource with an upper bound of directory execution. The proposed mechanism introduces SUSPEND state which is a state of execution resource. When the state of a process is SUSPEND, no CPU time is assigned to the process. Each execution resource has a counter of assigned CPU time. If the counter is equal to the usage limit number of the execution resource, the state of the execution resource is changed to SUSPEND state to limit the upper bound until the current time slice expires.

The process flows of the new process scheduler are depicted in Fig. 6 and Fig. 7. Fig. 6 shows the process flow of the process scheduler. The search performed by the process scheduler for the execution resource has the highest priority. If directory execution is selected in step (4) of the process (Fig. 6), the process scheduler searches the leaf executions of the directory execution in step (5).

Fig. 7 shows the process flow of the process scheduler for the case in which the directory execution is the execution resource with an upper bound. First, this algorithm selects a leaf execution resource should be executed in next time slot in step (1). Next, the counter assigned CPU time is incremented in step (2). If the counter is equal to usage-limit number, the state of the selected process changed to a SUSPEND state. If the counter is equal to time-slice number in step (5) or the execution state of the selected execution is SUSPEND, the algorithm searches an execution again. Otherwise, the process shown in Fig. 7 is completed successfully, then the *execid* of the selected execution is returned.

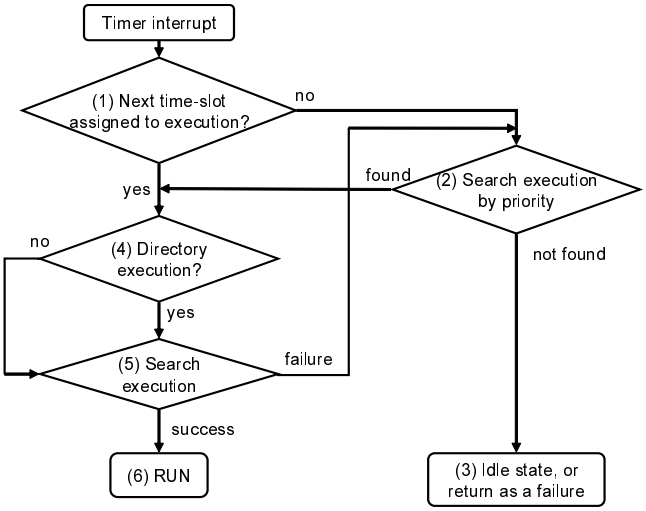


Fig. 6 Process flow of the process scheduler

3.3 Advantages

The proposed mechanism can prevent the occupation of a CPU resource for attacks such as DoS attacks by using an execution with an upper bound of CPU time. Because the proposed mechanism can control the upper bound of a directory execution resource, the abuse of CPU time of malicious processes can be efficiently restricted.

It is possible that the authority of a process is deprived by the attack from the outside. Assume that an execution with an upper bound is attached to all processes that connect to the network. In this case, even if the process is taken over by attackers, the influence of the DoS attack on the CPU resource can be controlled.

The specification of recent computers has been improved, and little software occupies the CPU at a high rate. Therefore, it is feasible to use execution with an upper bound for each process without degrading the service performance.

4 Implementation

4.1 ***Tender*** Operating System

The design of the proposed mechanism is based on the execution resource. The execution resource requires the separation of execution resource and process resource in an OS. However, existing OSes except for ***Tender***, do not separate the degree of CPU usage (execution resource) from a process. In contrast, because each resource can exist independently in ***Tender***, execution resource

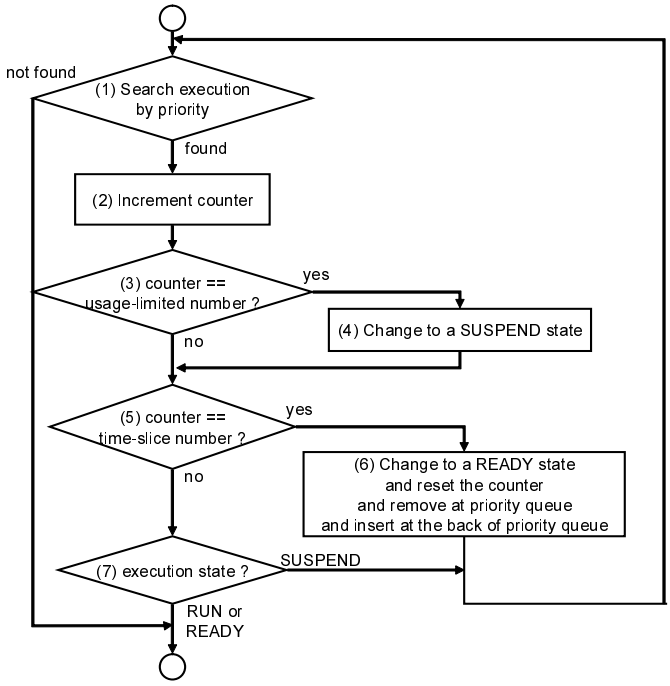


Fig. 7 Process flow when the directory execution with an upper bound is found

has been separated from process resource. For this reason, the proposed mechanism is implemented on the ***Tender*** operating system by using the execution resource.

In ***Tender***, OS resources can exist independently. The objects to be controlled and managed by ***Tender*** are known as "resources." Resources have identifiers and names for operations. The resource identifiers and resource names include location information that indicates the particular machine on which the OS is installed. Fig. 8 shows the structure of a resource identifier and resource name. Resource names have a tree structure. An example of a resource name is "/machine1/process/procA."

The interface for the operation of resources is a unified interface named the Resource Interface Controller (RIC). Programs that use resources are called through the RIC. The programs that use each resource are separated from each other. In addition, the sharing of a program component by multiple programs that use resources is not allowed. The program modules consist of five program components, namely, "open," "close," "read," "write," and "control." The RIC has a pointer table that includes all the pointers of the program components. Each program that manages resources has to call any program component through the RIC. Bypassing the RIC is prohibited in the ***Tender*** kernel.

There is a separate management table for each resource. In addition, pointers referring to values within the other resource management table are prohib-

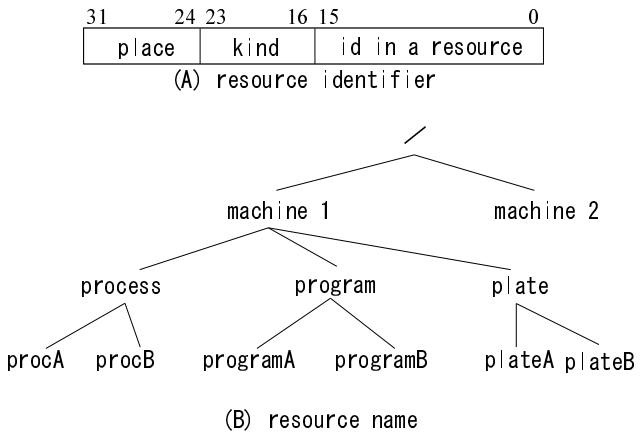


Fig. 8 Resource identifier and resource name

ited. The existence of an individual resource does not depend on the existence of other resources or processes because the management table for each resource is separate. In other words, the resources are independent of each other.

Since the individual resources are independent of each other, ***Tender*** is able to recycle them. Thus, ***Tender*** can preserve process resources instead of deleting them at process termination, and it can recycle the preserved process resources during process creation. Therefore, by preserving and recycling process resources, users can realize a reduction in the cost of process creation and termination [10]. In addition, an instant synchronous interprocess communication (ISIPC) mechanism is implemented in ***Tender***. The ISIPC mechanism can realize both instantaneous communication and synchronization of data[11].

4.2 Execution Resource

The proposed mechanism was implemented in the program component of execution resource and a process scheduler on ***Tender***. In order to manage the upper bound of directory execution, the information of assigned CPU time and upper bound of each directory execution resource were added to the management table of the execution resource. In addition, in order to designate the upper bound of a directory execution resource, the interfaces and function of creat_execution(mips) and control_execution(execid, mips) were modified. Table 2 shows the interfaces of these functions.

The process scheduler is invoked in every timer interrupt. The interval of the timer interrupt is 1 ms on ***Tender***. The process flow of process scheduler is shown in Fig. 6 before the leaf execution resource with upper bound and the directory execution resource with upper bound are introduced. The algorithm of the process scheduler for directory execution resource with an upper bound was implemented as follows. When the selected directory execution resource

Table 2 Modified operation interfaces of the execution resource

Form	Contents of operation
creat_execution (mips, upper bound)	Create the execution specified by mips with upper bound, and return the execution identifier execid. When mips is between 1 and 100 it signifies the performance regulation execution degree, when it is 0 or negative is signifies the priority of the execution degree.
control_execution (execid, mips, upper bound)	Change the execution degree of execid to mips with upper bound. mips is interpreted as in creat_execution.

in step (4) of Fig. 6 has an upper bound, the implemented algorithm shown in Fig. 7 is invoked in step (5) in Fig. 6. In this case, the algorithm checks whether the assigned CPU time of the directory execution resource is equal to the upper bound of it or not. If the assigned CPU time is equal the upper bound, the state of the directory execution resource is changed to SUSPEND state. The process scheduler searches directory execution resource and leaf execution resource in the execution tree recursively until it finds an execution resource should be assigned to next time slot.

The length of time slot, time block, and time slice on ***Tender*** are 1 ms, 100 ms, and 100 ms, respectively. Thus, one time block has 100 time slots. Next, the max number of process resource and execution resource are 79, and 255 respectively. In this case, the maximum number of child execution resource in a directory execution resource is less than 255.

5 Evaluation

5.1 Purpose of Evaluation

To show the feasibility of the proposed method, we evaluated the proposed mechanism on ***Tender***. We investigated whether the proposed method can control the upper bound of the CPU resources for the services.

Four evaluations were performed.

1. Evaluation for a general OS without the proposed mechanism shows how attack processes influence a program which is executed on a genral OS.
2. Basic evaluation shows how the proposed mechanism can control the performance of each process and limit the upper bound for process groups.
3. Evaluation for a case involving an attack shows how the proposed mechanism limits the influence of an attack service.
4. Evaluation using the Apache web server shows the feasibility of the proposed mechanism when a real application program is attacked.

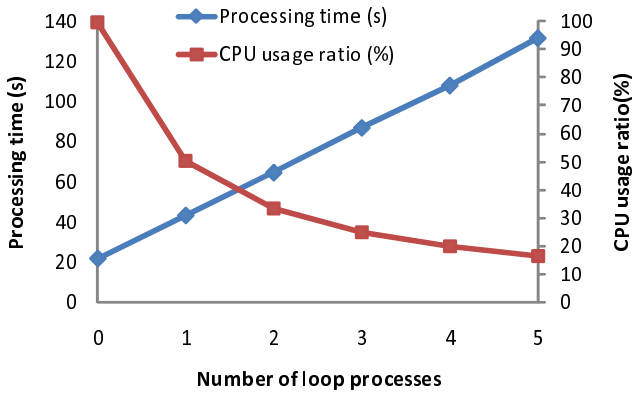


Fig. 9 Results of an imprecise computational program on a general OS

Table 3 Degree of execution resource in the basic evaluation

	Service A	Service B	Service C
case	exec 1	exec 2	exec 3
1	6	6, MAX 100%	6, MAX 100%
2	6	6, MAX 100%	6, MAX 75%
3	6	6, MAX 100%	6, MAX 50%
4	6	6, MAX 100%	6, MAX 25%
5	6	6, MAX 50%	6, MAX 50%
6	6	6, MAX 50%	6, MAX 25%

5.2 Evaluation for a general OS without the proposed mechanism

We evaluated the processing time and the CPU usage ratio of a program to show the influence of attacks for CPU resource in a general OS without the proposed mechanism. We used an imprecise computational program which calculates an approximate solution for natural logarithms. This program is CPU bound, thus it is importrant to prevent attack programs from using excessive CPU time. In addition, we executed some attack processes (loop process). The processing of eash process executes an infinite loop.

The evaluation was performed on a computer (CPU: Pentium4 3.0GHz, OS: FreeBSD 4.3-RELEASE). The imprecise computational program is executed when the number of loop process is from 0 to 5. We measured the processing time and CPU usage ratio of imprecise computational program.

Fig. 9 shows the results of the evaluation.

5.3 Basic Evaluation

An execution tree was constructed before the evaluation. This execution tree included three process groups (services A, B, and C). Each process group

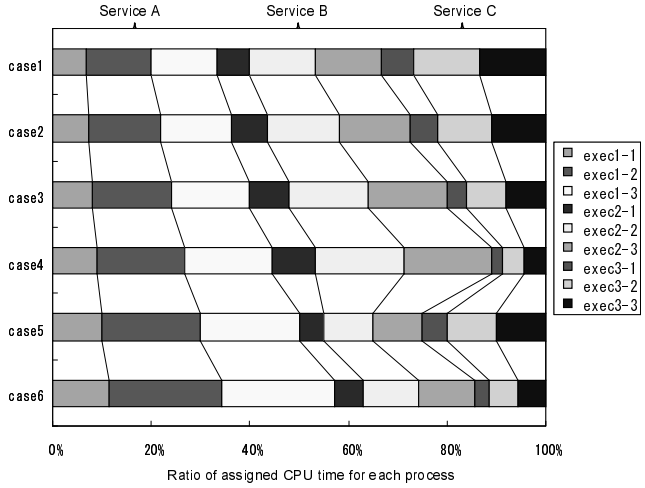


Fig. 10 Results of the basic evaluation

involved three processes. The processing of eash process executes an infinite loop. These processes were executed in a computer (CPU: Celeron 2.8GHz, OS: ***Tender***).

Table 3 shows the performance and priority of the execution resource of each process group in the execution tree. The execution resource (directory execution) of service A was given priority. The execution resources (directory executions) of services B and C were limited by an upper bound. The upper bounds of these execution resources were varied in this evaluation.

Fig. 10 shows the results of the basic evaluation. These results indicate that our proposed mechanism can control each process group according to the upper bound and that our proposed mechanism effectively limits the upper bound for process groups.

5.4 Evaluation for a Case Involving an Attack

5.4.1 Evaluation method

We evaluated the processing time of a normal service A (SA) and an attack service B (SB) to show the influence of an attack service on the proposed mechanism. SB tries to obtain as much CPU time as possible. Fig. 11 shows the execution tree used in this evaluation. SB was attached to the directory execution with an upper bound. The directory execution is used for limiting the upper bound of SB.

Two experiments were performed in the evaluation. In the first experiment, the number of processes in SB was changed from 1 to 5. The upper bound of SB was 50%. In the second experiment, the number of SB was 3. The upper bound

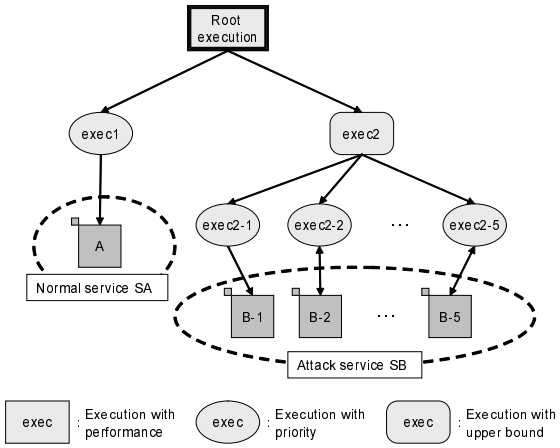


Fig. 11 Evaluation environment for a case involving an attack

of SB was changed from 25% to 100%. In both the experiments, the priority of the directory execution was changed. First, the priority of the directory execution (SB) was the same as that of SA. Next, the priority of the directory execution (SB) was higher than that of SA. The processing of a process that consists of SA is the same as that of the process that consists of SB. The processing executes an infinite loop. The experiments were performed using a computer (Processor: Celeron 2.8GHz, OS: ***Tender***)

5.4.2 Evaluation for Number of Attack Processes

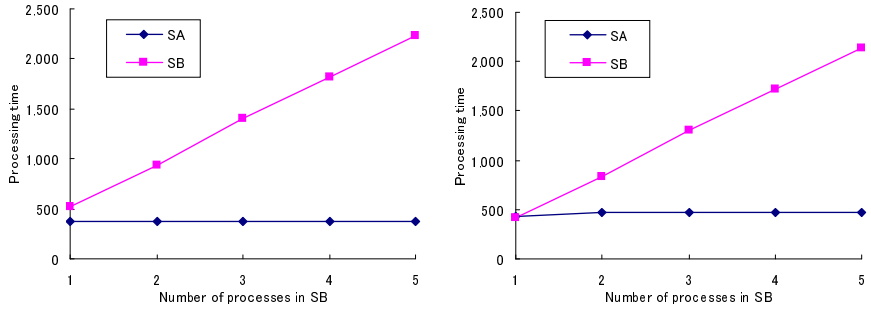
Fig. 12 shows the behavior of the processing time as the number of processes in SB changes. In the evaluation shown in Fig. 12 (A), the priority of SA is the same as that of SB. In the evaluation shown in Fig. 12 (B), the priority of SB is higher than that of SA. The processing time of each service is plotted on the y-axis, and the number of processes in SB is plotted on the x-axis.

In Fig. 12, the processing time of SA is constant because the upper bound of SB is restricted by the directory execution with an upper bound. These results show that the proposed mechanism can limit the influence of SB.

5.4.3 Evaluation for Number of the Upper Bound of Execution Resource

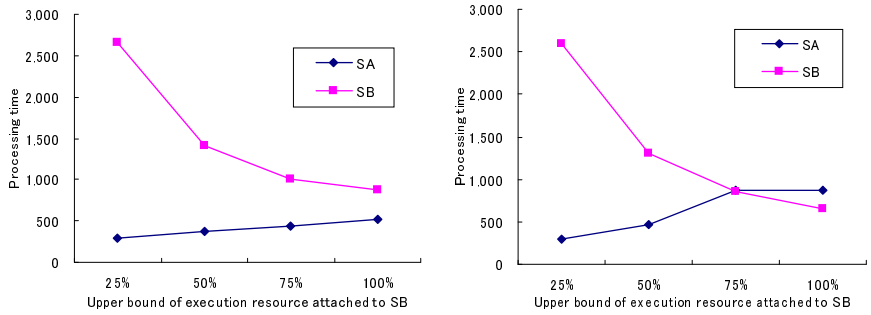
Fig. 13 shows the processing time of each service for the case in which the upper bound of the execution resource attached to SB is changed. The upper bound was increased from 25% to 100%. In the evaluation shown in Fig. 13 (A), the priority of SA is the same as that of SB. In the evaluation shown in Fig. 13 (B), the priority of SB is higher than that of SA.

In Fig. 13, as the proposed mechanism restricted the deterioration in the performance of SB, the processing time of SA decreased. These results show



(A) Priority: SA = SB (B) Priority: SB > SA

Fig. 12 Behavior of processing time as the number of processes in SB changes



(A) Priority: SA = SB (B) Priority: SB > SA

Fig. 13 Processing time when the upper bound of the execution resource attached to SB is changed

that the proposed mechanism can restrict CPU abuse caused by a malicious service. In addition, the performance of SA was improved because the assignment of CPU time for SB was restricted.

From Fig. 13 (B), the processing time of SA is found to be longer than that of SA shown in 12 (A). These results show that SA was influenced by SB. In addition, the processing time of SA is constant when the upper bound is more than 75% and the processing time of SB is decreased. In this case, the processing of SB finishes first, and thus, the entire CPU time is assigned to SA. Thus, the processing time of SA did not increase.

Table 4 Evaluation environment

	Server machine	Client machine
OS	***Tender***	Windows Vista
Processor	Celeron D 2.8GHz	Core 2 Duo E6600 2.4GHz
Memory	768MB	2048MB
Program	Apache HTTP Server 1.3.33	ApacheBench 2.2.17.0

5.5 Evaluation Using Apache Web Server

5.5.1 Evaluation Environment and Method

In order to evaluate the proposed mechanism, we simulated a DoS attack against a Web server. The scenario was as follows. We assumed that an attacker intrudes into a server machine and takes control of processes. The attacker creates processes and attempts to consume the CPU time in the machine. To consume the CPU time, the attacker runs processes (attack processes) that execute an infinite loop. As a result, the performance of the web server (Apache) is influenced by the attack.

Table 4 shows the evaluation environment. To measure the server performance, a client program was run on a client machine. The average of the response time of the client program was calculated as a measure of the performance. A web page on the server machine was accessed 100 times by the client program. The two machines are connected with 100Base-TX ethernet.

The evaluations were performed for three cases. First, the evaluation was performed without applying the proposed mechanism to programs. Second, the evaluation was performed by applying the earlier proposed mechanism (leaf execution resource with an upper bound) to programs. Third, the evaluation was performed by applying the proposed mechanism (directory execution resource with an upper bound) to programs.

In the first evaluation, the Apache web server and attack processes were executed by attaching execution with priority to the processes. In the second and third evaluations, an execution tree was constructed, and then, the Apache web server and attack processes were executed. In these evaluations, the response time of the Apache web server was measured.

Two experiments were performed under the three conditions. Experiment 1 was performed as follows.

1. The number of attack processes was changed from 1 to 5.
2. The evaluation was performed for two cases of the priority of the web server. In the first case, the priority of the Apache web server was the same as that of the attack processes. In the second case, the priority of attacker processes was higher than that of the Apache web server.
3. The upper bound of the execution resource attached to attack processes was 50%.

Experiment 2 was performed as follows.

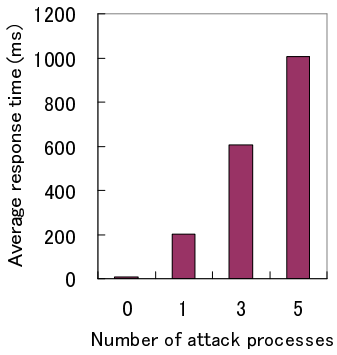


Fig. 14 Evaluation without the proposed mechanism (priority is same)

1. The number of attack processes was 3.
2. The evaluation was performed for two cases of the priority of the web server. In the first case, the priority of the Apache web server was the same as that of the attack processes. In the second case, the priority of attacker processes was higher than that of the Apache web server.
3. The upper bound of the execution resource attached to attack processes was changed from 10% to 100%.

5.5.2 Evaluation without the Proposed Mechanism

The evaluation described in this subsection was performed without applying the proposed mechanism to programs. In this evaluation, the Apache web server and attack processes were executed by attaching execution with priority. As described in section 5.5.1, there are two conditions for the priority: the priority of the processes in Apache is the same as that of the attack processes or the priority of the attack processes is higher than that of the processes in Apache.

Fig. 14 shows the results of experiment 1 for the case in which the priority of SA is the same as that of SB. The figure indicates that the influence of the attack processes increased according to the number of attack processes. The average response time increased by 200 ms for each attack process. As a result, the increase in the number of attack processes has a significant influence on the average response time.

In addition, the average response time was measured when no attack process exists. The result was 6.9 ms. This corresponds to the case in which the number of processes is 0, as shown in Fig. 14.

When the priority of the attack processes is high, the response of the Apache web server was timeout. This result implies that attack processes consumed the CPU time. Thus, the processes of Apache were not executed. Hence,

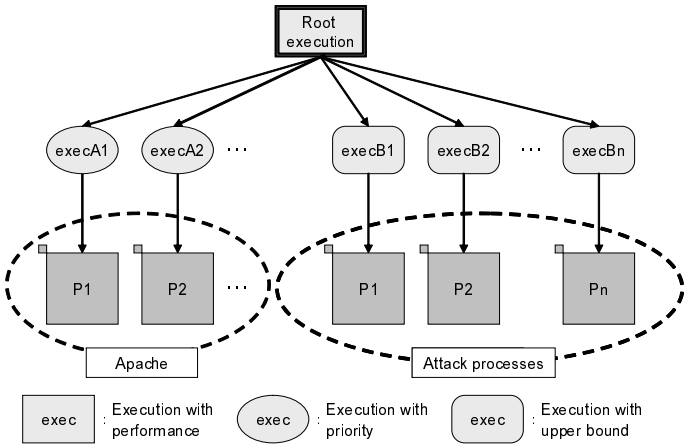


Fig. 15 Execution tree for evaluating leaf execution resource with an upper bound

in this case, an increase in the number of attack processes has a significant influence on the service of Apache.

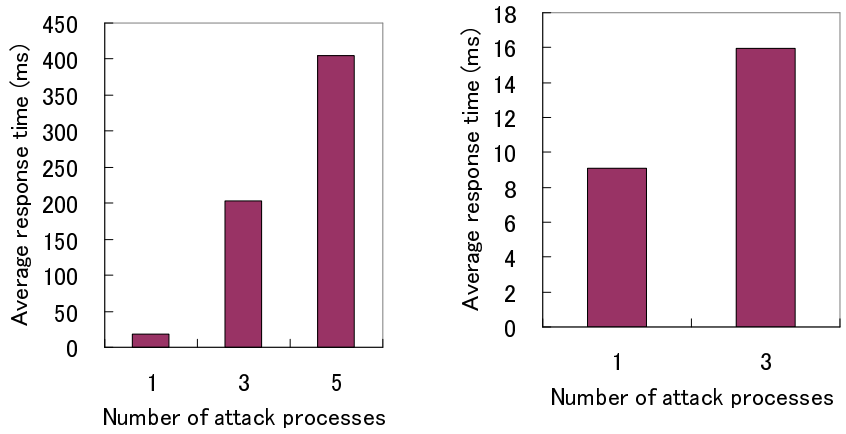
5.5.3 Evaluation with the Leaf Execution Resource

The evaluation described in this section was performed by applying the earlier proposed mechanism to programs. Leaf executions that have an upper bound were attached to the attack processes. Experiments 1 and 2 were performed. Fig. 15 shows that an execution tree is used for this evaluation.

Fig. 16 shows the results of Experiment 1. In Fig. 16 (A), the upper bound is 50%, and in Fig. 16 (B), the upper bound is 20%. In both cases, the number of attack processes is 1, 3, and 5.

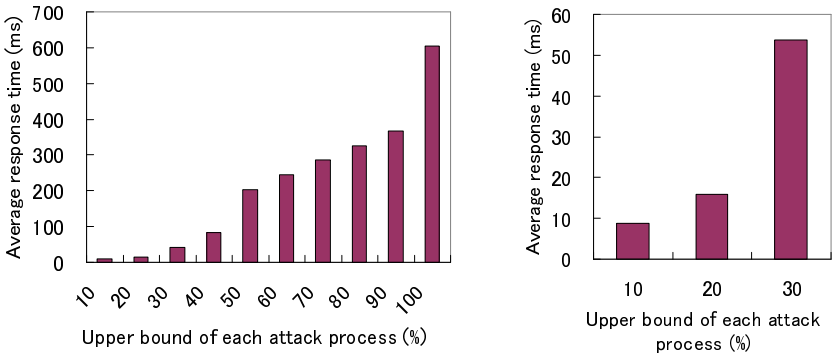
In Fig. 16 (A), the influence of the attack processes increased with the number of attack processes. When the priority of attack processes is higher than that of Apache and the number of attack processes is more than 3, the response of Apache was timeout. When the number of attack processes was 1, the average response time was 14.5 ms. Thus, Fig. 16 (B) depicts the results for the case in which the upper bound of the attack process is 20%. These results show that if the priority of the attack processes is high, the influence is significant. When the proposed mechanism was not applied to attack process, the response time could not be measured. However, in this case, the response time can be measured. These results show that the leaf execution resource with an upper bound can limit the influence of attack processes.

Fig. 17 shows the results of experiment 2. As shown in Fig. 17 (A), the proposed mechanism can limit the influence of the attack processes when the upper bound of the attack processes is less than 40%. In addition, the limitation of the influence of attack processes is small when the upper bound of the attack processes ranges from 50% to 90%. However, the proposed mech-



(A) Priority: SA = SB, Upper bound=50% (B) Priority: SB > SA, Upper bound=20%

Fig. 16 Evaluation results of experiment 1 for evaluation of leaf execution



(A) Priority: SA = SB (B) Priority: SB > SA

Fig. 17 Evaluation results of experiment 2 for leaf execution evaluation

anism can limit some influence in this case. When the upper bound of the attack processes is 100%, the influence of the attack processes for Apache is significant, because the proposed mechanism does not restrict the influence of attack processes.

Fig. 17 (B) shows that the response of the Apache web server was timeout when the upper bound of the attack processes was higher than 40%, because the priority of the attack processes was higher than that of Apache processes. This result shows that the leaf execution with an upper bound can restrict the influence of the attack processes when the upper bound is low.

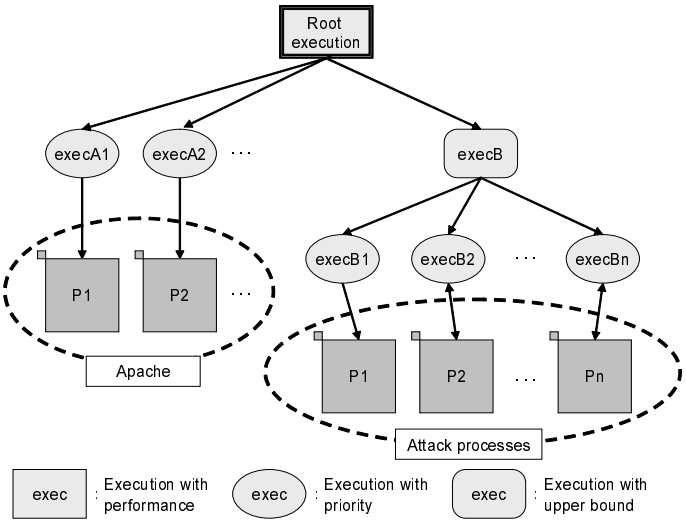


Fig. 18 Execution tree for evaluating directory execution resource with an upper bound

5.5.4 Evaluation with the Directory Execution Resource

Fig. 18 shows the execution tree used in this evaluation. Each attack process was attached to leaf execution, and then each leaf executions was attached to the directory execution resource with an upper bound. As a result, the proposed mechanism can restrict the upper bound of the attack processes as a unit of the process group.

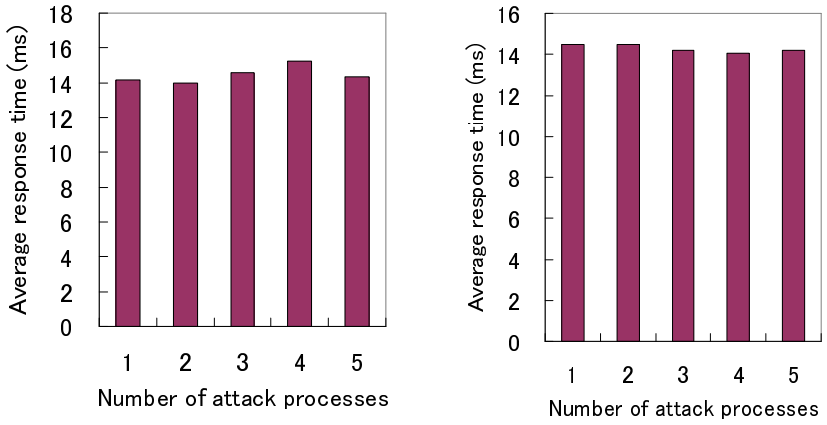
As described in section 5.5.1, two experiments were performed.

Fig. 19 shows the results of experiment 1. Figs. 19 (A) and (B) show that the number of attack processes has little influence on the average response time of the Apache web server. Therefore, the proposed mechanism can restrict the decrease in the performance of the Apache web server.

Figs. 20 (A) and (B) show that the proposed mechanism can restrict the influence of the attack processes for the Apache web server when the upper bound of the attack processes ranges from 10% to 80%. However, when the upper bound is 90%, the influence of the attack processes is comparatively high.

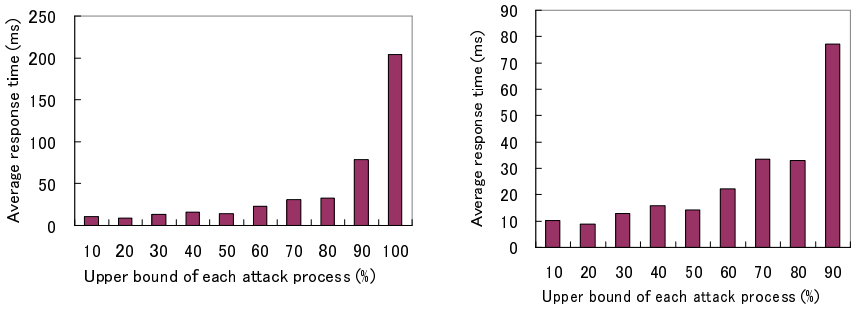
5.6 Summary of Evaluations

As described in section 5.1, we performed three evaluations. Table 5 summarizes the results of three evaluations. Table 5 shows that our proposed mechanism can limit the influence of attack processes and improve the performance of normal services.



(A) Priority: SA = SB (B) Priority: SB > SA

Fig. 19 Evaluation results of experiment 1 for evaluation of directory execution



(A) Priority: SA = SB (B) Priority: SB > SA

Fig. 20 Evaluation results of experiment 2 for evaluation of directory execution

6 Related Work

Most defense techniques against Internet-originated DoS attacks have targeted the transport and network layers of the TCP/IP protocol stack [12]. Our research focuses on the access control mechanism of and the rate limiting technique for CPU resources.

In the past decade, resource accounting techniques and resource protection techniques for defending against DoS attacks have been proposed, and these techniques have been successfully utilized to counter DoS attacks. The Scout operating system has an accounting mechanism for all the resources employed by each I/O path [13]. Scout also has a separate protection domain for each

Table 5 Summary of Evaluations

Evaluations	Summary of evaluation results
Basic Evaluation	1. Our proposed mechanism can control each process group according to the upper bound. 2. Our proposed mechanism effectively limits the upper bound for process groups.
A Case Involving an Attack	1. Our proposed mechanism can limit the influence of an attack service (SB) because the upper bound of SB is restricted. 2. The performance of a normal service (SA) is improved because the assignment of CPU time for SB is restricted.
Using Apache Web Server	1. The leaf execution with an upper bound can restrict the influence of the attack processes when the upper bound is low. 2. The directory execution with an upper bound can restrict the influence of the attack processes for the Apache web server when the upper bound of the attack processes ranges from 10% to 80%. When the upper bound is 90%, the influence of the attack processes is comparatively high.

path. The present research focuses on the I/O paths and not on the access control of CPU resources.

The resource control mechanism is implemented by the Resource Control Lists (RCL) in Linux RK[14]. This mechanism has resource reserve function and rate limiting function of CPU time, and this mechanism controls a process as a unit of RCL and enforces the RCL rules to it. Thus, this mechanism cannot control CPU time of users as a unit of rate limiting. In addition, this mechanism is not suitable for the type enforcement mechanism in secure OS, because the subject of RCL does not match the subject of secure OS.

Resource containers [15] have been proposed, and they can be used to account for and limit the usage of kernel memory. This container mechanism supports a multilevel scheduling policy; however, it only supports fixed-share scheduling and regular time-shared scheduling. Resource reservation framework for MINIX 3 has been proposed[16]. This framework provides limited hard real-time and soft real-time support for applications. In addition, this frame work improves dependability by enabling temporally isolated execution in order to prevent DoS attacks.

Execution resources with upper bounds are classified under resource accounting techniques [17]. The execution resource can control the maximum extent of CPU usage of programs for preventing abuse of CPU resources. The policy of rate limiting can be enforced for a CPU resource by using an access control mechanism for the execution resource. In addition, the proposed access control model can be applied to general OSes and secure OSes.

7 Conclusion

We proposed an improved mechanism for achieving a bound on the execution performance of process groups in order to limit unnecessary processor use. We improved the previously proposed mechanism used for controlling the upper bound for a process. We introduced directory execution as an execution resource with an upper bound. In order to do so, the process scheduling mechanism was changed for the control of an execution resource with an upper bound of directory execution. The proposed mechanism was implemented in execution resource on ***Tender*** operating system.

To show the feasibility of the proposed method, we evaluated the proposed mechanism on ***Tender***. The results of the basic evaluation show that our proposed mechanism can control each process group according to the upper bound. Further, the results of evaluation for a case involving an attack show that the proposed mechanism can limit the infuence of SB because the upper bound of SB is restricted by the directory execution with an upper bound. Finally, the results of the evaluation conducted the Apache web server show that the directory execution resource with an upper bound can efficiently limit the influence of the attack processes for Apache processes by restricting the upper bound of the attack processes.

References

1. CERT/CC Statistics 1988-2005: http://www.cert.org/stats/
2. Wagner D, Dean D: Intrusion Detection via Static Analysis, In Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp.156 − 168 (2001)
3. Hofmeyr S A, Forrest S, Somayaji A: Intrusion Detection using Sequences of System Calls, Journal of Computer Security, Vol.6, No.3, pp.151 − 180 (1998)
4. Sekar R, Bendre M, Bollineni P, Dhurjati D: A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors, In Proceedings of of IEEE Symposium on Security and Privacy, pp. 144–155 (2001)
5. Sandhu R S, Coyne E J, Feinstein H L, Youman C E: "Role-Based Access Control Models," IEEE Computer, vol. 29, no. 2, pp. 38–47 (1996)
6. Security-Enhanced Linux, http://www.nsa.gov/selinux/
7. Tabata T, Nomura Y, Taniguchi H: A Mechanism of Regulating Execution Performance for Process Group by Execution Resource on ***Tender*** Operating System, Systems and Computers in Japan, Vol. 38, No. 4, pp. 63-78 (2007).
8. Tabata T, Hakomori S, Yokoyama K, Taniguchi H: A CPU Usage Control Mechanism for Processes with Execution Resource for Mitigating CPU DoS Attack, International Journal of Smart Home, vol. 1, no. 2, pp. 109–128 (2007)
9. ***Tender*** project, http://www.swlab.cs.okayama-u.ac.jp/lab/tani/research/tender-e.html
10. Tabata T, Taniguchi H: An Improved Recyclable Resource Management Method for Fast Process Creation and Reduced Memory Consumption, International Journal of Hybrid Information Technology (IJHIT), vol. 1, no. 1, pp.31-44 (2008)
11. Yamauchi T, Fukutomi K, Taniguchi H: ISIPC: Instant Synchronous Interprocess Communication, Journal of Next Generation Information Technology, vol.1, no.3, pp.75-83 (2010)
12. Garg A, Reddy A: Mitigation of DoS attacks through QoS regulation, In Proceedings of IEEE International Workshop on Quality of Service (IWQoS), pp.45–53 (2002)
13. Spatscheck O, Petersen L L: Defending Against Denial of Service Attacks in Scout, In Proceedings of 3rd Symp. on Operating Systems Design and Implementation, pp. 59–72 (1999)

14. Miyoshi A, Rajkumar R: Protecting Resources with Resource Control Lists, In Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01), pp. 85–94 (2001)
15. Banga G, Druschel P, Mogul J C: Resource containers: A new facility for resource management in server systems, In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99), pp. 45–58 (1999)
16. Mancina A, Faggioli D, Lipari G, Herder J N, Gras B, Tanenbaum A S: Enhancing a dependable multiserver operating system with temporal protection via resource reservations, Real-Time Systems, vol. 43, no. 2, pp.177–210 (2009)
17. Mirkovic J, Reiher P: A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM Comput. Commun. Rev., vol. 34, no. 2, pp. 39–53 (2004)