A Mechanism that Bounds Execution Performance for Process Group for Mitigating CPU Abuse

Toshihiro Yamauchi, Takayuki Hara, and Hideo Taniguchi

Graduate School of Natural Science and Technology, Okayama University,
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
{yamauchi,tani}@cs.okayama-u.ac.jp,hara@swlab.cs.okayama-u.ac.jp

Abstract. 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. For preventing the abuse of CPU resources, we had earlier proposed a new type of execution resource that controls the maximum CPU usage [5, 6]. 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

1 Introduction

The number of computers connected to 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 in the number of incidents of software vulnerability can be attributed to the widespread use of automated attack tools and the increasing number of attacks against systems connected to the Internet [1]. Therefore, various defense mechanisms against such attacks have been studied extensively, and these studies have gained a lot of attention.

Various defense mechanisms include firewalls, an Intrusion Detection System (IDS) [2]; buffer overflow protection and access control mechanisms such as Mandatory Access Control (MAC) and Role Based Access Control (RBAC) [3]; and secure OS are examples of such defense mechanisms.

The secure OS [4] has been the focus of several studies. In particular, Security-Enhanced Linux (SELinux) has become of major interest. Even if the authority is taken, secure OS makes the range of the influence a minimum. However, the CPU resource, which is an important resource for executing a program, is not the object of the access control. 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.

In an earlier study, we proposed a new type of execution resource that controls the maximum CPU usage such that the abuse of CPU resources can be prevented [5, 6]. In order to prevent the abuse of the CPU resources, we propose an execution resource that can limit the upper bound of CPU usage. The previously proposed mechanism can control only one process at a time. Because, most of services 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 execution performance of a process group, in order to limit unnecessary processor use. The proposed mechanism is based on a previously proposed mechanism. The proposed mechanism introduces execution tree which deploy the upper bound of execution resource for the nodes of execution tree.

2 Execution Resource

In this section, we explain the concept of execution resource on the basis of the presentation in previous papers [5, 6].

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 example, 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 has been named execution resource. Therefore, only the execution resource involves the degree of CPU usage, and a process does not have a degree of CPU usage. Prior to the introduction of the execution resources, processes are listed on a linked list on the basis of their priority. After the introduction of the execution resources, it is these execution resources that 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.

The execution manager points to an execution with the highest degree of CPU usage. All processes need to be linked to executions 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.

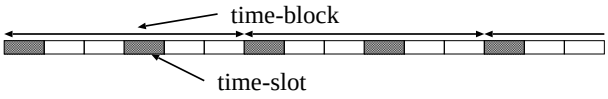

Fig. 1. Time slots and a time block

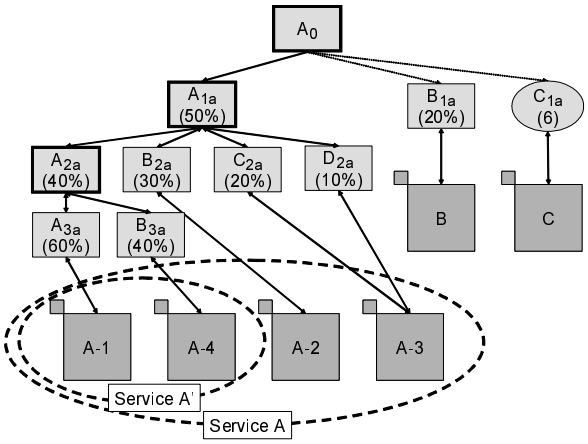

Fig. 2. Relationships between a process group and executions

Execution with performance Execution with performance includes a degree of CPU usage that indicates the proportion of bare processor performance. The bare processor performance can be defined 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 named a unit of CPU usage as a "time slot." We termed a group of time slots as a "time block." Fig. 1 shows the relation between time slots and a time block. An execution in which the degree of CPU usage is n% is assigned n% of the time slots in a time block.

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. 2 shows the relationships between a process group and

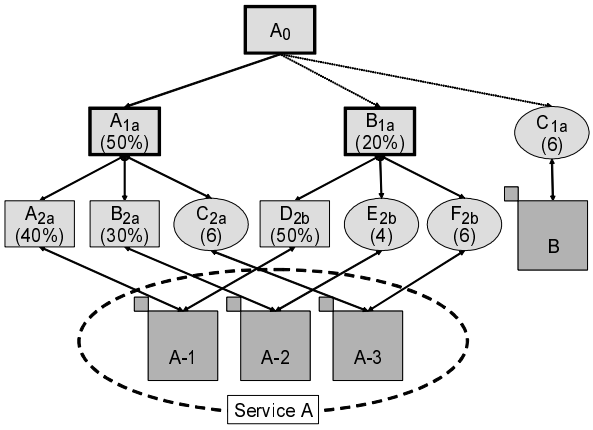

Fig. 3. Two process group executions

executions. 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 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 a ratio (%) to assigned CPU time of parent directory execution. In the leaf execution, the ratio corresponds to the point where the parent directory execution is defined 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. 3 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}, F_{2b}) have to be created and linked to each process (A, B, C) 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 8 operation interfaces for the execution resource for constructing an execution tree and controlling a program execution. Table 1 shows the interfaces.

3 Execution Resource with Upper Bound

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

In 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

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, pid)	Associate the execution execid and the process pid.
detach_execution (execid, pid)	Remove the association between execution execid and process pid.
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.

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, the performance of the service deteriorates significantly.

To prevent the abuse of the CPU resources, we propose an execution resource that helps to achieve an upper bound for the CPU usage ratio. In this execution resource, the CPU time is allocated according to the priority until the usage reaches a specified ratio in a time slice. 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 [5, 6], we can guarantee that the important processes will be carried out effectively by using the execution resources with a good performance.

3.2 Execution resource with upper bound for process group

In the previous mechanism, the execution resource with an upper bound was a leaf execution. Thus, the previous mechanism could be used only to control a process. We introduce directory execution as an execution resource with an

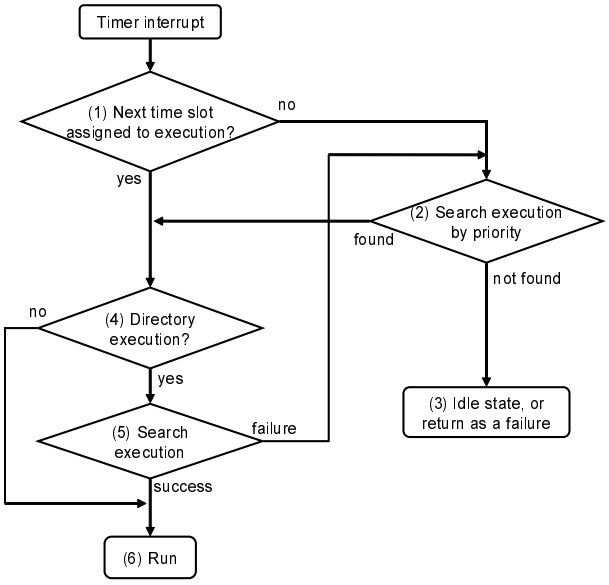

Fig. 4. Process flow of the process scheduler

upper bound. In order to do so, the process scheduler was changed for the control of an execution resource with upper bound of directory execution.

The process flow of the new process scheduler is depicted in Fig. 4 and Fig. 5. Fig. 4 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. 4), the process scheduler searches the leaf executions of the directory execution. Fig. 5 shows the process flow of the process scheduler when the directory execution is the execution resource with an upper bound. If a leaf execution resource is assigned a CPU time slot, the process illustrated in Fig. 5 is successfully completed.

4 Evaluation

We investigated whether the proposed method can control upper bound of the CPU resources for the services. We performed a basic evaluation and an evaluation for a case involving an attack.

4.1 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 involved three processes. Table 2 shows the performance and priority of execution

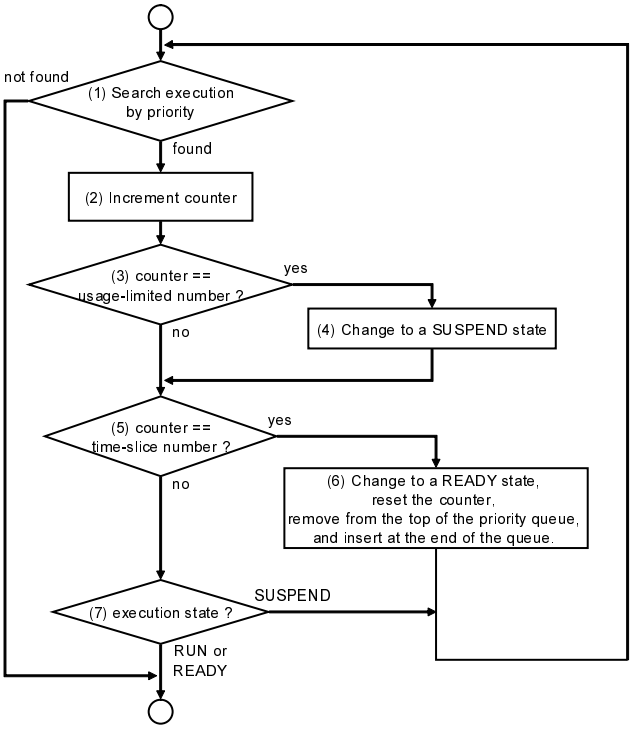

Fig. 5. Process flow when the directory execution with an upper bound is found

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 service B and C were limited by an upper bound. The upper bounds of these execution resources were varied in this evaluation.

Fig. 6 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.

4.2 Evaluation for a Case Involving an Attack

We evaluated the processing time of a normal service A (SA) and an attack service B (SB). SB tries to obtain as much CPU time as possible. In this evaluation, SB was attached to the directory execution with an upper bound. Fig. 7 shows the behavior of the processing time as the number of processes in SB changes. 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. The processing time of SA is constant because the upper bound of SB is restricted by the directory execution with an upper bound.

Table 2. 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%

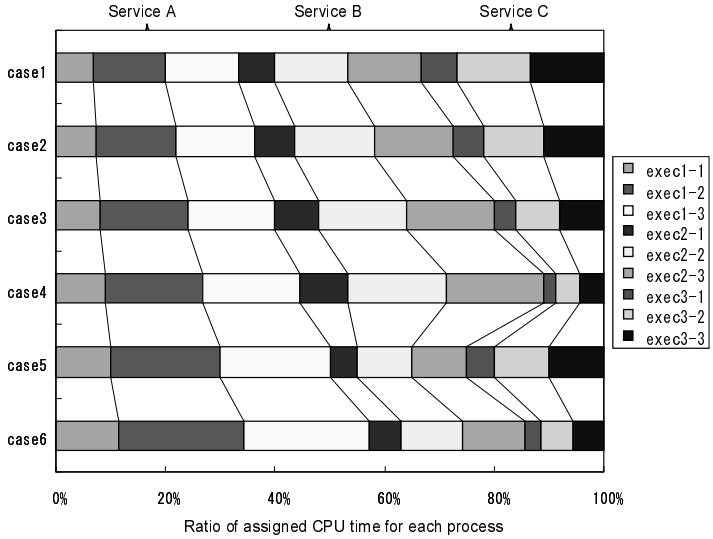

Fig. 6. Results of the basic evaluation

Fig. 8 shows the processing time of each service when the upper bound of the execution resource attached to SB is changed. The upper bound was increased from 25% to 100%. As the proposed mechanism restricted the deterioration in the performance of SB, the processing time of SA decreased. These results show that the proposed mechanism can restrict CPU abuse caused by a malicious service.

5 Related Work

Most defense techniques against Internet-originated DoS attacks have targeted the transport and network layers of the TCP/IP protocol stack [7]. Our research focuses on the access control mechanism of and the rate limiting technique for CPU resources.

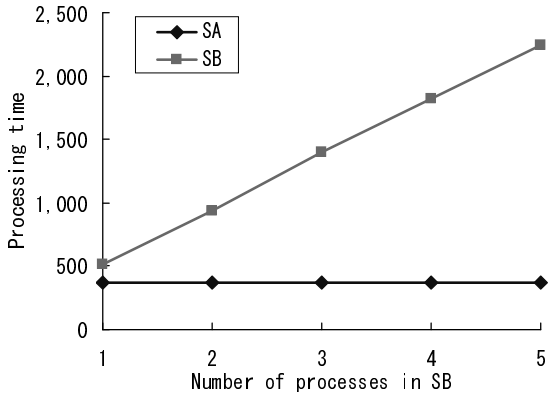

Fig. 7. Behavior of processing time as the number of processes in SB changes

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 [8]. Scout also has a separate protection domain for each path. The present research focuses on the I/O paths and not on the access control of CPU resources.

Resource containers [9] 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.

Execution resources with upper bounds are classified under resource accounting techniques [10]. 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.

6 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 scheduler was changed for the control of an execution resource with upper bound of directory execution.

The results of evaluations show that our proposed mechanism can control each process group according to the upper bound. These results also show that

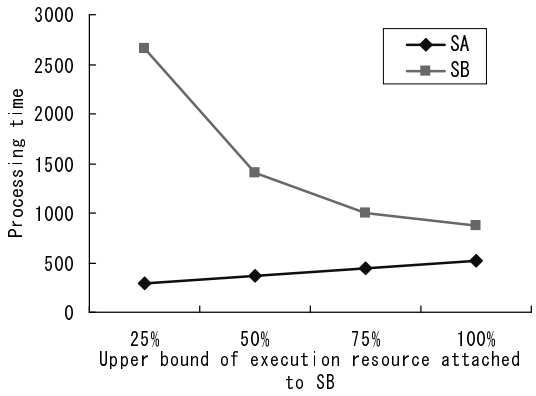

Fig. 8. Processing time when the upper bound of the execution resource attached to SB is changed

our proposed mechanism is effective in limiting the upper bound for the process groups.

References

1. "CERT/CC Statistics 1988-2005," http://www.cert.org/stats/
2. R. Sekar, M. Bendre, P. Bollineni and D. Dhurjati, "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors," Proc. of IEEE Symposium on Security and Privacy, pp. 144–155, (2001)
3. R. S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, "Role-Based Access Control Models," IEEE Computer, vol. 29, no. 2, pp. 38–47, (1996)
4. "Security-Enhanced Linux," http://www.nsa.gov/selinux/
5. T. Tabata, S. Hakomori, K. Yokoyama, H. Taniguchi, "Controlling CPU Usage for Processes with Execution Resource for Mitigating CPU DoS Attack," In: 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), pp. 141–146, (2007)
6. T. Tabata, S. Hakomori, K. Yokoyama, H. Taniguchi, "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)
7. A. Garg and A. Reddy, "Mitigation of DoS attacks through QoS regulation," In: IEEE International Workshop on Quality of Service (IWQoS), pp.45–53, (2002)
8. O. Spatscheck and L. L. Petersen, "Defending Against Denial of Service Attacks in Scout," In: 3rd Symp. on Operating Systems Design and Implementation, pp. 59–72 (1999)
9. G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," In: the Third Symposium on Operating Systems Design and Implementation (OSDI '99), pp. 45–58 (1999)
10. J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," ACM SIGCOMM Comput. Commun. Rev., vol. 34, no. 2, pp. 39–53 (2004)