An Improved Recyclable Resource Management Method for Fast Process Creation and Reduced Memory Consumption

Toshihiro Tabata and Hideo Taniguchi
Graduate School of Natural Science and Technology, Okayama University
{tabata, tani}@cs.okayama-u.ac.jp

Abstract

Processes can be executed concurrently in operating systems; they can be created dynamically and then terminate, i.e. process creation and termination are required for program execution. However the costs involved in process creation and termination make this procedure expensive. This procedure expensive, thus degrading the performance of program execution. To solve this problem, a fast process creation and termination mechanism is proposed. This mechanism is implemented by recycling process resources. In order to improve the efficiency of recycling, the management of preserved process resources for recycling is an important factor.

In this paper, we propose an improved resource management method for recycling process resources and an adaptive control mechanism. In the method, only one process resource with a program image is preserved for each program that occurs with high frequency of program execution. The proposed method can reduce the amount of memory consumption for preserved process resources in a concurrent execution environment. We also describe the implementation of the proposed method on the ***Tender*** *operating system and report the results of our experiments.*

1. Introduction

A process is a program that is being executed. Processes can be executed concurrently in Operating Systems (OS), and are created dynamically and then terminate; i.e. process creation and termination are required for program execution. However, the procedure of process creation and process termination are expensive, thus degrading the performance of program execution. We believe that reducing the cost of process creation and termination is a crucial issue for program execution in OS. For example, the Apache HTTP Server sometimes creates processes for executing CGI programs. In this case, the number of processes that are created is in proportion to the number of times they are accessed. Apache 1.3 is also a pre-forking server [1], [2]. Apache always has some idle processes to satisfy the MinSpareServers setting. Server settings such as the MinSpareServers have an effect on the benchmark results. However, this application-level approach is only applied to processes of Apache, i.e. the pre-forking mechanism only creates processes on Apache, and it cannot be applied to other programs such as CGI programs invoked by the Apache server. Therefore, a new method that can reduce the cost of process creation and termination for all programs is required at OS-level, since an OS-level approach could be applied to all programs.

In paper [3], a fast process creation and termination mechanism using recycling process resources is proposed and this mechanism is implemented on The ENduring operating system for Distributed EnviRonment (***Tender***) [4]. In ***Tender***, OS resources can exist independently, and ***Tender*** can also preserve used resources for recycling when a process terminates. This recycling mechanism can make the processing involved in process creation and termination

much faster. However, preserved resources for recycling consume a large amount of memory. Therefore, these preserved process resources have to be managed efficiently.

In order to improve the efficiency of recycling, the authors of [3] proposed an efficient resource management method for recycling process resources that focuses on the frequency with which they are utilized in program execution [5]. In this method, there are two types of preserved resources for recycling from the standpoint of frequency of use during program execution. For programs that have a high frequency of program execution, the process resources are preserved with program image. Thus, the process resources can be recycled for processes that are associated with the same program image. When programs have a low frequency of execution, their process resources are preserved without program image. Thus, they can be recycled for any process that satisfies the recycling condition.

The previous proposed method is a simple mechanism and just focuses on the frequency of program execution. Thus, the previous proposed method can reduce the amount of memory consumed by preserved resources, but it is insufficient to just reduce this factor. Many programs are executed concurrently in a real system, i.e. we also have to consider the concurrency of process execution. The costs become much higher when some processes associated with a same program image are executed concurrently. In addition, the previous proposed method was not adaptive for the change of the frequency of program execution.

In this paper, we propose an improved resource management method for recycling process resources that focuses on concurrent process execution. We introduce adaptive-control mechanism in the resource management method. In the proposed method, only one process resource with a program image is preserved for each program that is executed frequently. When a preserved process resource associated with the same program image as a terminated process exists, the process is broken up into process resources without a program image. The proposed method can reduce the amount of memory consumed by selecting the type of preserved process resources. It can also improve efficiency when recycling process resources in a concurrent execution environment. Thus, it can reduce the cost of process creation. The adaptive-control mechanism that is introduced in the proposed method monitors the number of program execution, and dynamically decides which programs should be preserved with a program image. We also describe the implementation of the proposed method on the ***Tender*** operating system and report the results of our experiments.

2. Related Work

Threads are a way for a program to split itself into two or more tasks that run simultaneously. Threads are 'lightweight', but the protection between threads is weak because they share the same memory space. Therefore, they are undesirable from the viewpoint of security. In addition, the debugging of multi-thread programming is difficult. An ideal solution would be the implementation of processes that are lightweight in a similar way to threads.

Many researches topics for reducing the cost of process creation have been proposed in the past. “Sticky bit functions” and the “vfork system call” approach can reduce the cost of process-creation in UNIX [6]. “Sticky bit” is a research topic based on fast process creation that enables the OS to recycle a region of text. “Demand paging” and “Copy on Write” (CoW) are also research areas that focus on memory management. Demand paging and CoW can also reduce the cost of process creation.

Because OS resources have high modularity in ***Tender***, each OS resource can exist independently. This mechanism can realize a mechanism for recycling process resources. However, the existing OS cannot recycle any process resources and cannot preserve process resources for recycling. The ***Tender*** operating system differs from the existing OS in that process resources can be recycled during the processing used for process creation.

Furthermore, research has also been undertaken to study page control and cache memory in virtual memory [7], [8], [9]. These research techniques manage a memory region as an operation unit for memory management, e.g. an operation unit is a physical page of memory. This research is similar to resource recycle from the standpoint of controlling memory resources.

3. *Tender* Operating System

3.1. Separation and Individualization of Resources

In ***Tender***, objects to be controlled and managed by the operating system are known as "resource". Resources are given resource identifiers and resource names for operations. Resource identifiers and resource names include location information that indicates a particular machine. Resource names are managed by tree structure. An example of resource name is "/machine1/process/procA".

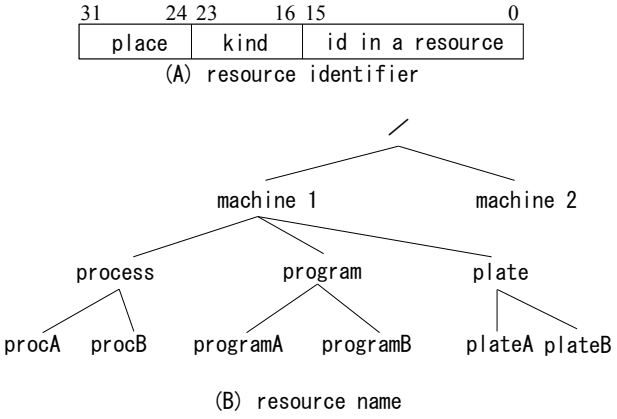

Figure 1. Resource identifier and resource name

The interface for the operation of resources is unified. Programs that operate resources are called through a unified interface. Thus, functions can call the programs that operate both local resources and remote resources through the unified interface. The programs that operate each resource are separated between those resources. In addition, shared programs are eliminated. The program modules consist of 5 program components "open", "close", "read", "write" and "control" The unified interface is named Resource Interface Controller (RIC). RIC has a pointer table that has all pointers of program components. Each resource management must call RIC in order to call any program components. Bypassing RIC is prohibited in ***Tender*** kernel.

The management table for each resource is also separated between each resource. In addition, pointers between the management tables of each resource are prohibited. The

existence of an individual resource does not depend on other resources including processes, because the management table for each resource is separated. That is, each resource can exist irrespective of the existence of other resources.

3.2. Process Resources

A process is composed of various process components. The process components in memory space of ***Tender*** are called "process resources".

Figure 2 shows the memory resources on ***Tender***. "Virtual region" is a resource that virtualizes the data storage region, which is mapped to physical memory or external storage. It contains information about the storage area, which is in physical memory or in external storage, in its management table. "Virtual space" is a space for the virtual address and corresponds to the mapping table, where a virtual address is mapped to a physical address. "Virtual user space" is a space that is accessible from the processor by the virtual address. It is created by attaching the "virtual region" to "virtual space" and is deleted by detaching. Here, 'attaching' means to store the information in the data storage region as an entry in the mapping table.

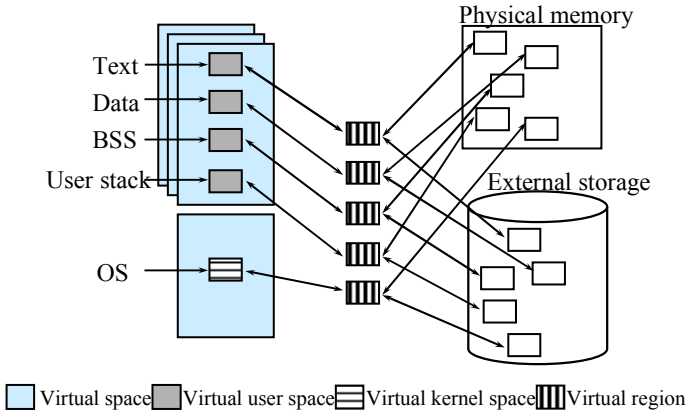

Figure 2. Resources of process and memory on *Tender*

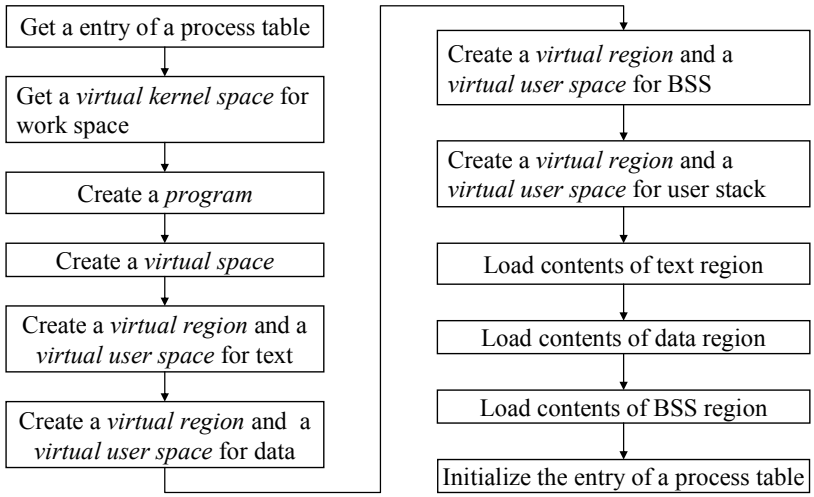

Figure 3. Flowchart for process creation

3.3. Process Creation

Figure 3 shows the flowchart of process creation on ***Tender***. The processing consists of twelve steps.

4. Recycling of Process Resources

4.1. Overview

In the existing OS, the existence of process resources depends on the process that owns them, because the management information for each resource is stored in the process management table. If a process terminates, its entry in the process management table is cleared. As a result, its process resources and any information regarding the process resources are cleared, too.

Separation and individualization of resources enables ***Tender*** to preserve process resources for recycling. By using this mechanism, ***Tender*** can preserve process resources instead of deleting them at process termination. As a result, ***Tender*** can recycle preserved process resources during process creation.

Figure 4 and Figure 5 show the flow for process creation and termination. ***Tender*** enables process resources to be preserved at process termination. This means that, in cases where users want to release a resource, users can preserve it without actually deleting it. In addition, it enables process resources to be prepared during process creation. This means that in cases where users wish to obtain a resource, we can recycle a resource that has been preserved in advance without creating it again. Therefore, users can realize a reduction in the cost of process creation and termination by preserving and recycling process resources.

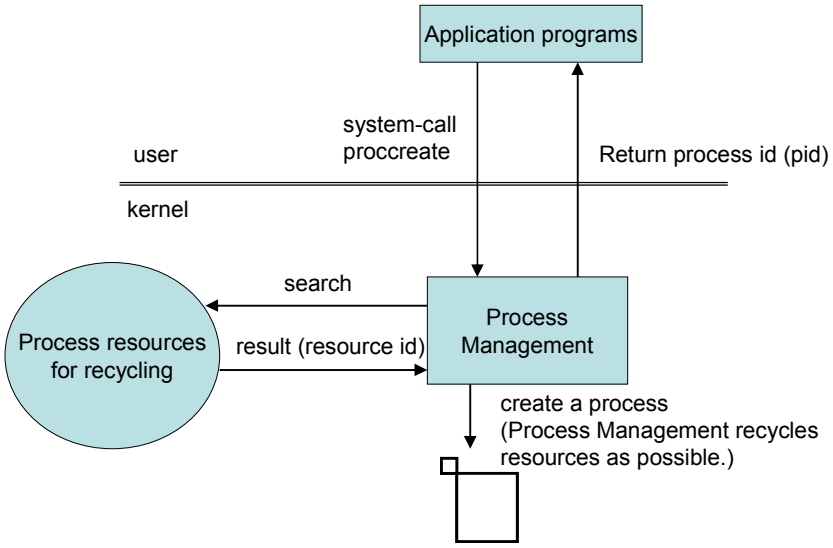

Figure 4. Flow for process creation by recycling process resources

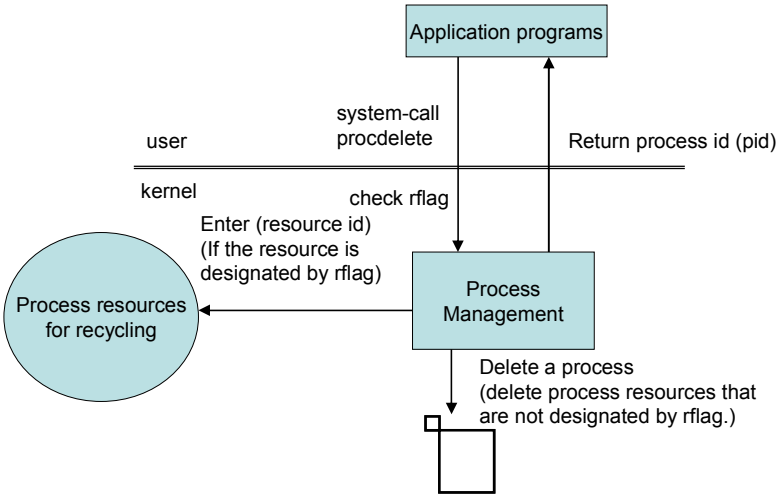

Figure 5. Flow for process termination by preserving process resources

Table 1. Types of preserved process resources for recycling

Type	Condition of recycling
Preserved process resources with program image (RDP)	Type of program
Preserved process resources without program image (RIP)	Virtual space: always Virtual region: size of virtual region

4.2. Two Types of Preserved Resources

An efficient resource management technique for recycling process resources is previously proposed [5]. This technique focuses on frequency in program execution. Preserved resources are classified into two types of preserved resources for recycling from the standpoint of the frequency of program execution. Table 1 shows these two types of preserved resources.

For programs with a high frequency of program execution, process resources are preserved with a program image. This type of preserved resource is named RDP (Recyclable resource that is Dependent on Program). RDP can be recycled when a process is created from the same program. RDP consists of virtual memory, i.e. a memory image of the program in virtual memory space. The memory image includes the text region, the data region, the BSS region and the stack region in the memory space. The contents of each region are loaded into the memory space.

For programs with a low frequency of program execution, process resources are preserved without a program image. This type of preserved resource is named RIP (Recyclable resource that is Independent of Program). RIP consists of virtual memory and a virtual region. The process resources for programs that are used with low frequency are broken up into virtual memory and unmapped memory objects (virtual regions). There is no memory object in the preserved virtual memory. The virtual region consists of physical memory and some on-disk area and a management table for mapping between physical memory and external storage. Because the virtual memory and virtual regions have no information associated with a particular program image, they can be recycled for every process creation step if they satisfy

the following conditions; virtual memory can be recycled for every process creation event; virtual regions can be recycled if the size of the virtual region is the same as the requested size.

5. Proposed Method

5.1. Problem

The method proposed in paper [5] can reduce the amount of memory consumed by preserved resources. However, a method that focuses on the frequency of program execution is unable to reduce memory use sufficiently because costs become much worse when some processes in a program are executed concurrently. Because many programs are executed concurrently in a real system, we need to consider the concurrency of process execution.

In the present system, a process that has a high frequency of program execution is preserved as RDP, i.e. RDP can be recycled when a process is created from the same program. As a result, when a process is created from other programs, the preserved process resources cannot be recycled. In this method, the number of RDP for a particular program is greater than 1. One block of RDP is always recycled when the program is executed. However, the remaining blocks of RDP (more than 2) are not recycled if plural processes in the program are not executed concurrently. Therefore, the remainder may be redundant and could consume a significant amount of memory. Namely, in a case where the efficiency of recycling of RDP becomes worse, the cost of process creation and termination increases. In addition, the memory consumption of preserved process resources for recycling increases, because RDP contains program images. Moreover, even if two or more blocks of RDP exist, the effect of recycling the RDP is relatively small because of text-sharing.

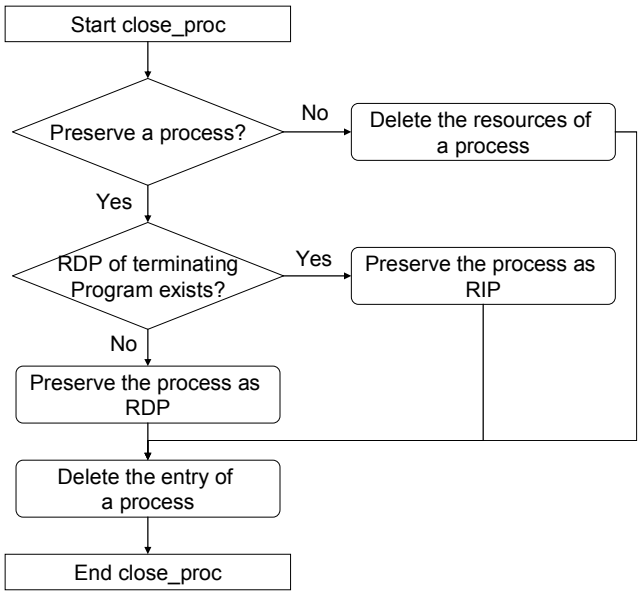

Figure 6. Flow chart of process termination in the proposed method

5.2. Improved Resource Management

To solve the problem described above, only one RDP block is preserved for each program used with high frequency during program execution. When a preserved block of RDP for the same program as a terminating process exists, the process is preserved as RIP.

Figure 6 shows a flow chart of the proposed method. First, the OS decides whether it should preserve resources from a terminating process. Next, the OS checks whether an RDP resource for the terminating program exists. If the RDP exists, the OS preserves the process as RIP. If not, the OS preserves the process as RDP.

Advantages:

(１) Resources preserved as RIP increase, because resources preserved as RDP are limited to 1 in the proposed method. The ratio of the success of recycling to process creation can be improved, because resources preserved as RIP can be recycled for all programs during process creation. As a result, the processing involved in process creation and termination becomes faster.

(２) The percentage of recycled resources is improved by increasing the amount of process resources preserved as RIP. Thus, the proposed method requires far fewer preserved resources for recycling than are needed in the original method. In addition, the proposed method is much more effective for fast process creation and termination in spite of requiring less memory for preserved resources.

The price of memory chips has decreased enormously recently, and the amount of memory installed in a typical PC is sometimes over 1GB. Because large amounts of memory are now usually installed in computers, it is not difficult to prepare sufficient memory for preserving process resources. Therefore, there is still a lot of memory in computers that remains unused, and we can utilize this unused memory to improve the performance of the computer. Our proposal can provide efficient usage of unused memory; it utilizes unused memory to preserve resources relating to processes, which can then be recycled for fast process creation, i.e. we can convert unused memory into processing power. We expect that our proposal will improve the response time of services that involve many process creation and termination steps. Program images are usually smaller than image files and movie files. Although our proposal does not require a lot of memory, it can realize fast process creation and termination.

5.3. Adaptive control function

Various programs are simultaneously executed in a system. The kinds of programs are dynamically changed according to the requests of users. Thus, the proposed system should periodically control the configuration of recycle. We call this mechanism adaptive control function. As mentioned above, all of program calls pass through RIC. Thus, RIC can intercept all requests of process creation. RIC logs history of process creation and feedbacks to process management. Because process management can know which process is executed frequently, it can decide the type of reserved resources for each process.

6. Evaluation

To show the practicality of the proposed method, we evaluated it on the ***Tender*** operating system. We implemented the proposed method in the ***Tender***, and then used the benchmark program. The benchmark program invokes a process creation system-call and a process

termination system-call according to a recorded real program execution sequence. We recorded the sequence on the web server in our lab. The sequence has a length of 1000.

The experiments were performed on a PC (CPU: Pentium III 750MHz, OS: ***Tender***). In the experiments, we measured the memory consumption of the preserved resources and the processing time of the benchmark program. The benchmark program repeats the process creation and termination steps, i.e. the processing time represents the processing time for process creation and termination.

We performed experiments using the proposed method and the original method as proposed in paper [5]. The number of existing processes was changed from 1 to 20. The parameters for the existing processes mean that the benchmark program creates a process and terminates a process to maintain the number of existing processes during the experiments. We measured both the memory consumption and processing time at intervals of 100 program executions. Figure 7, Figure 8, Figure 9 and Figure 10 show the results of the experiments.

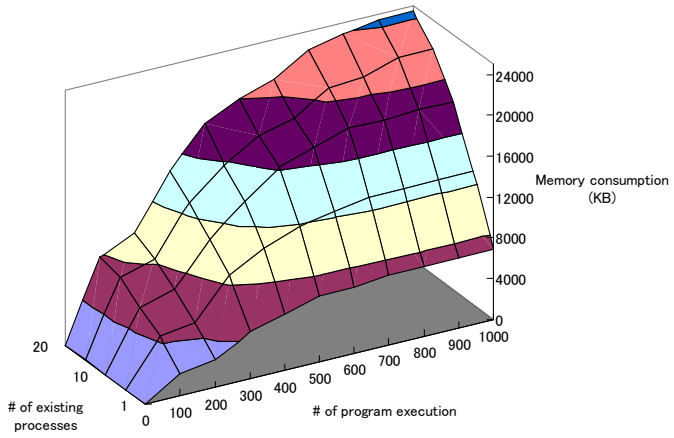

Figure 7. Memory consumption of the original method

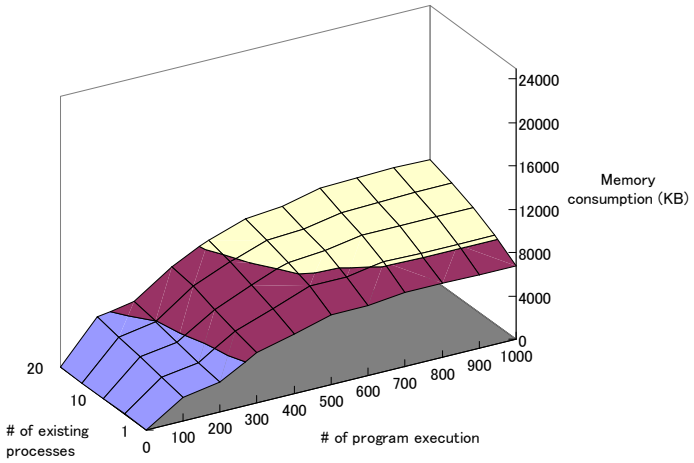

Figure 8. Memory consumption of the proposed method

The proposed method can reduce the amount of memory consumed for recycling, i.e. the amount of memory consumed by the proposed method is much less than that required by the original. This is because the percentage of recycled resources in the proposal is higher than in the original. As a result, the proposed technique does not need to pool excessive recyclable resources to improve the performance of process creation and termination.

In particular, the memory consumption of the original method increases in proportion to the existing processes. On the other hand, the increase in memory consumption is much less in our proposal because it improves the percentage of recycled resources. That is, the proposal can reduce the amount of preserved resources that are not recycled.

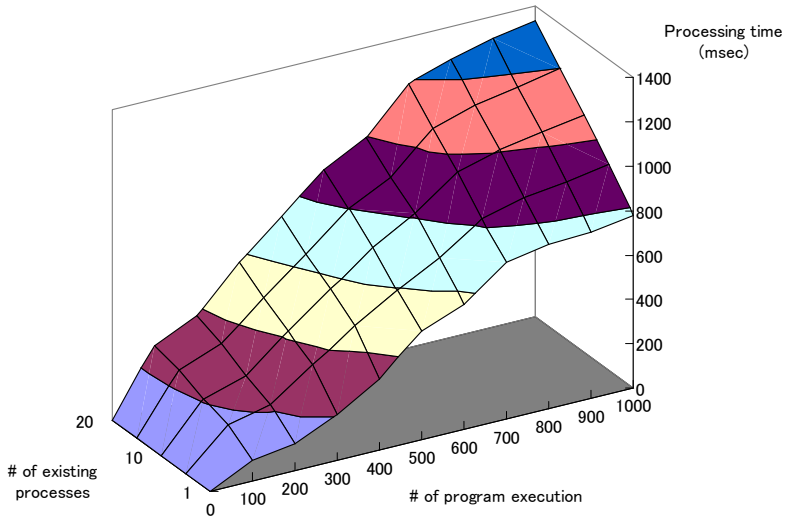

Figure 9. Processing time of the original method

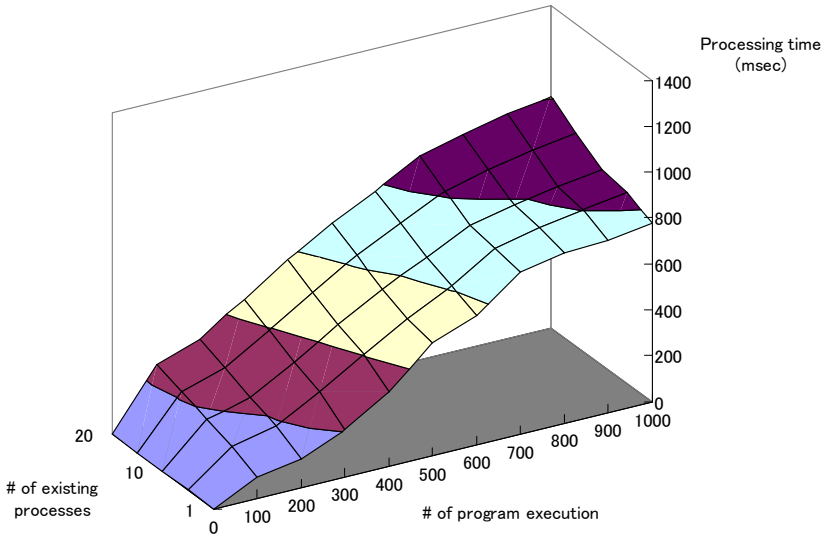

Figure 10. Processing time of the proposed method

Total processing time with the proposed method is hardly influenced by increases in the number of existing processes. On the other hand, processing time in the case of the original technique is influenced when the number of existing processes increases, because of the low ratio of recycled resources. As a result, the OS has to create new resources for process creation.

For between 700 and 1000 executions of the program, memory consumption with the proposed method shows only a small increase in preserved resources increases. We call this duration "a steady state". The steady state comes earlier with the proposed method than with the original, because the proposal needs less memory than the original. Thus, it needs less steps of program execution. In addition, processing with the proposed method is faster than that with original, particularly during the steady state. The cause of this difference is the percentage of recycled resources. In the original, memory consumption increases for the steady state. In this case, the preserved resources cannot satisfy most of the resources requested resources for process creation. As a result, the OS has to create new resources for process creation.

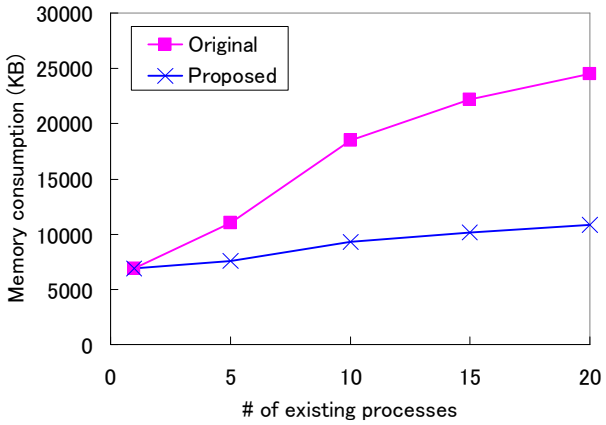

Figure 11. Memory consumption (number of program execution is 1000)

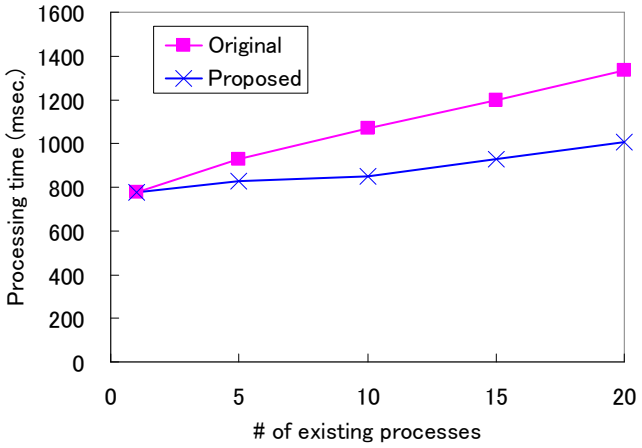

Figure 12. Processing time (number of program execution is 1000)

Next, Figure 11 and Figure 12 show the results when the number of program executions is 1000. These figures show that the proposal is faster by about 33% than the original. In addition, the proposal can reduce memory consumption by about 56% compared with the original. These results also show the effectiveness of the proposal.

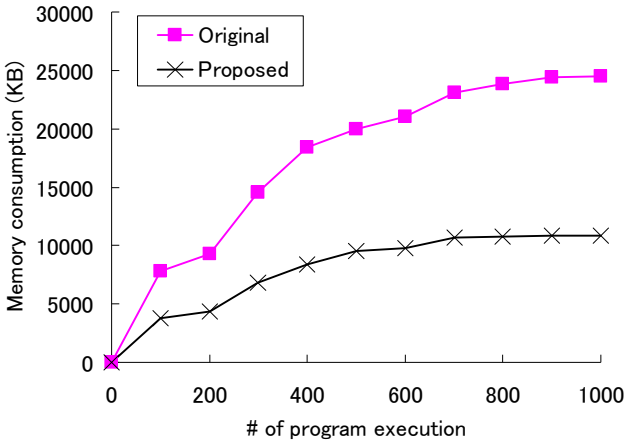

Figure 13. Memory consumption (number of existing processes is 20)

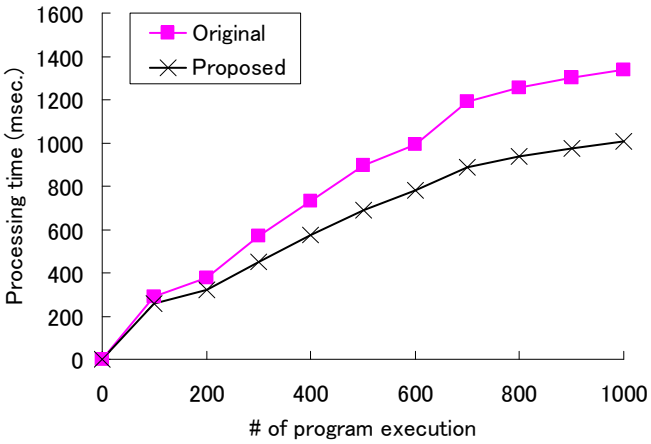

Figure 14. Processing time (number of existing processes is 20)

Figure 13 and Figure 14 show the result when the number of existing processes is 20. These results show that the proposal is about 27% faster than the original. In addition, the proposal can reduce memory consumption by about 54% compared with the original. These results also show the effectiveness of the proposal.

7. Conclusion

We have proposed an improved resource management method for recycling process resources. The proposed method can make the implementation of process creation and termination faster, and it can reduce the memory consumption required for preserving resources for recycling. Only one process resource with a program image is preserved for each program that occurs with high frequency of program execution. When a preserved

process resource with a program image for the same program as a terminating process exists, the process is broken up into process resources without a program image. The ratio of the success of recycling to process creation can be improved, and the recycling rate of the resources is improved by preserved resources without a program image.

We also report the results of our evaluation of the proposed method. The proposed method can reduce the memory that is consumed for recycling. The amount of memory required for recycling is less than half of that needed for our original technique. Processing using our new proposal is faster than that of the original technique. In addition, the proposal suffers little influence from increases in the number of existing processes. The proposed method shows good performance in terms of process creation and termination from the standpoints of processing time and the amount of memory consumed.

Acknowledgement: This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B) 18300010.

References

[1] Dean Gaudet, "Apache Performance Notes," http://httpd.apache.org/docs/misc/perf-tuning.html

[2] "The Apache Modeling Project," http://f-m-c.org/projects/apache/html/Apache_Modeling_Project.html

[3] H. Taniguchi, Y. Aoki, M. Goto, D. Murakami, T. Tabata, "***Tender*** Operating System based on Mechanism of Resource Independence," IPSJ Journal, Vol. 41, No. 12, pp. 3363-3374, 2000. (in Japanese)

[4] "***Tender*** project," http://www.swlab.it.okayama-u.ac.jp/lab/tani/research/tender-e.html

[5] T. Tabata, H. Taniguchi, "Proposal of Efficient Resource Management for Recycling Process Elements," IPSJ Transactions on Advanced Computing Systems, Vol. 44, No. SIG 10(ACS2), pp. 48-61, 2003. (in Japanese)

[6] J. S. Quarterman and A. Silberschatz and J. L. Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System," ACM Computing Surveys, Vol. 17, No. 4, pp. 379-418, 1985.

[7] A. Braunstein, M. Riley, J. Wilkes, "Improving the Efficiency of UNIX File Buffer Caches," SOSP: 12th ACM Symposium on Operating Systems Principles, pp. 71-82, 1989.

[8] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, "Informed Prefetching and Caching," SOSP'95: 15th ACM Symposium on Operating Systems Principles, pp. 79-95, 1995.

[9] V. S. Pai, P. Druschel, W. Zwaenepoel, "IO-Lite: A Unified I/O Buffering and Caching System," OSDI'99: USENIX Association 3rd Symposium, pp. 15-28, 1999.

Authors

Toshihiro Tabata

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

Hideo Taniguchi

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