Proposal and Implementation of Heterogeneous Virtual Storage Coexisted of Single Virtual Storage and Multiple Virtual Storage

Toshihiro TABATA
Faculty of Information Science and Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

and

Hideo TANIGUCHI
Faculty of Engineering, Okayama University
3-1-1 Tsushimanaka, Okayama 700-8530, Japan

ABSTRACT

Most of Operating Systems (OSs) provide processes with virtual memory. One advantage of this technique is that programs can be larger than physical memory. In addition, this technique abstracts main memory into large address space and frees programmers from the limitation of main memory. Single Virtual Storage (SVS) or Multiple Virtual Storage (MVS) are implemented in current OSs, but SVS and MVS do not coexist in existing OSs. If they coexist in an operating system, users can make use of each advantage. In this paper, we propose Heterogeneous Virtual Storage (HVS). Because SVS and MVS can coexist in HVS, HVS can provide both of the advantages of SVS and MVS to users. We also describe about implementation of HVS on The ENduring operating system for Distributed EnviRonment (***Tender***). After that, we explain contents of experiments and report that result.

Keywords: Virtual storage, Operating system, HVS, Process creation, Process migration

1 Introduction

Most of Operating Systems (OSs) provide processes with virtual memory. One advantage of this technique is that programs can be larger than physical memory. In addition, this technique abstracts main memory into large address space and frees programmers from the limitation of main memory.

Single Virtual Storage (SVS) and Multiple Virtual Storage (MVS) are typical examples of virtual memory. One feature of SVS provides a single virtual address space to all processes. On the other hand, one feature of MVS is that each process has an independent virtual address space. SVS or MVS are implemented in current OSs, but SVS and MVS do not coexist in existing OSs. If they coexist in an OS, users can make use of each advantage.

In this paper, we propose Heterogeneous Virtual Storage (HVS). Because SVS and MVS can coexist in HVS, HVS can provide both of the advantages of SVS and MVS to users. We also describe about implementation of HVS on The ENduring operating system for Distributed EnviRonment (***Tender***)[1]. ***Tender*** has been developed at Kyushu University and Okayama University in Japan. After that, we explain contents of experiments and report that result.

2 Related Work

Memory protection is one of the main issues of SVS, and several studies have been made on it. Opal[2] is an OS that provides a single virtual address space. It presents the concept of memory sharing and protection and realizes protection domain. Opal threads execute within protection domain in a single virtual address space and there is no loss of protection.

In UNIX, MVS is implemented, and shared memory is used for data sharing. Efficient memory operation method is also proposed for remote machine in distributed environments[3].

In HVS, a process which needs memory protection has own virtual address space. It is similar to MVS. On the other hand, processes which do not need memory protection, exist on a virtual address space. It is similar to SVS. In order to cooperate with each other efficiently, these processes exist in a virtual address space, and trust in each other.

The protection mechanism of HVS restricts process creation and migration. To create a process on an existing virtual address space and to migrate a process, the access right is needed.

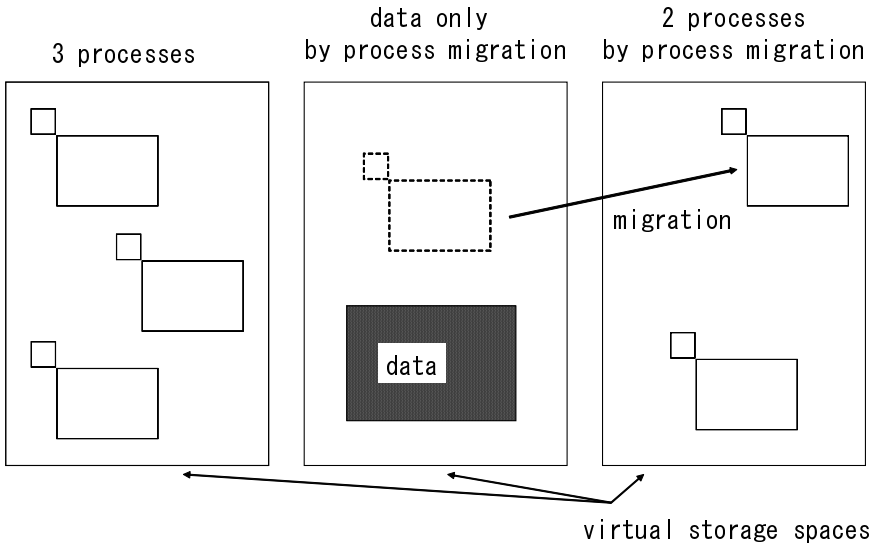

Figure 1. Overview of HVS.

3 Heterogeneous Virtual Storage

Comparison with Single Virtual Storage and Multiple Virtual Storage

In this section, we compare SVS with MVS.

SVS provides a single virtual address space to all processes. Therefore, processes share the virtual address space, but a process cannot use all of virtual address space. In addition, it is necessary to protect memory space of each process from other processes. The feature of MVS is each process has an independent virtual memory space. The address space is separated from other processes logically. Thus, each process can own virtual address space.

Memory protection is one of the main issues of SVS, but memory protection mechanism is not necessary for MVS, because each process has own virtual address space. Besides, MVS is superior in available address space, because each process has own address space. On the other hand, SVS is superior in context switch, because it does not include change of an address space.

Current OSs are implemented SVS or MVS, but SVS and MVS do not coexist in an existing OS. If they coexist in OSs, users can make use of each advantage. The purpose of this study is to propose new virtual storage that has advantages of SVS and MVS.

Overview of HVS

MVS and SVS have contractive advantages, as stated above. If MVS and SVS can coexist in an OS, we can make use of each advantages. Therefore, we propose Heterogeneous Virtual Storage coexisted of MVS and SVS.

Figure 1 shows the overview of HVS. There are three functions of HVS.

(1) HVS provides multiple virtual address spaces. In figure 1, three virtual address spaces exist.

(2) There are more than zero processes on a virtual space.
In figure 1, no process exists on the center of virtual address spaces, after a process migrated. The virtual address space is for data.

(3) A process can migrate to other virtual address spaces. In figure 1, a process migrates from the center of virtual address spaces to the right one.

Therefore, HVS has both SVS's advantages and MVS's advantages.

By using function 1 and function 2, we can make use of both advantages only. Besides, we can make use of these advantages effectively by using function 3. For example, processes which cooperate with each other closely can be executed in the same virtual address space. On the other hand, processes which do not cooperate with each other, can be located in each virtual address space separately. As a result, these processes are protected from other processes. If the relation of processes changes dynamically, a process can migrate to other virtual address space.

A virtual address space where data only exist can be created on HVS. Hence, multiple processes can share a huge data in a virtual address space, and can use it with time-sharing.

4 Implementation on *Tender*

We implemented HVS in ***Tender***. In this section, we explain ***Tender*** and describe about the prob-

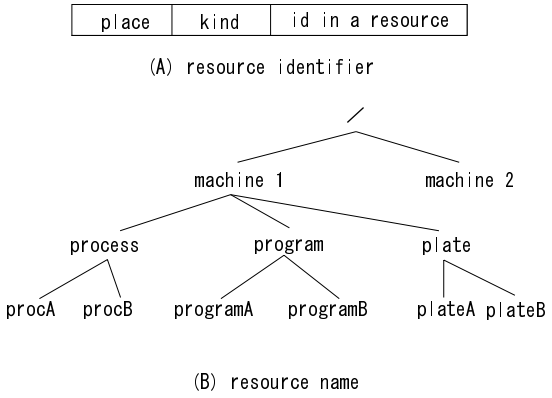

Figure 2. Resource identifier and resource name.

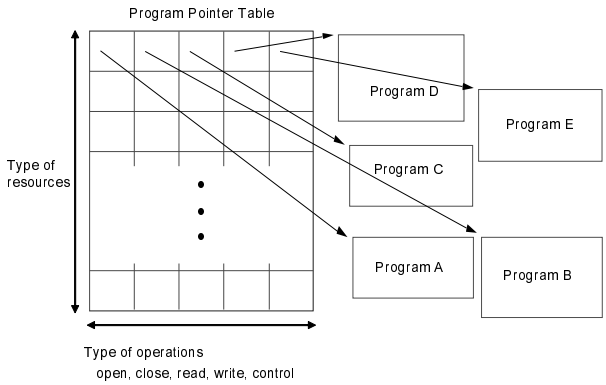

Figure 3. Program pointer table.

lems of the implementation, and the solution.

Overview of *Tender*

Resource Independent: *Tender* is based on resource independent. Resource independent means that the objects that OS operates are separated as resources and are individualized. The resources are given resource name and resource identifier. Figure 2 shows the structure of resource identifier and resource name. Besides, the interface of the operation of resources is unified. Furthermore, the components of programs that operate each resource are separated. The management information of each resource is also separated between each resource.

On existing OS, existence of process elements depends on process that own process elements, because the management information of each process element is stored in the process management table. If a process is deleted, the entry of process management table is cleared. As a result, process elements and the information of process elements are also cleared.

On ***Tender*** OS, the existence of each resource does not depend on other resources and process, because the table of each resource is separated. Therefore, processes and virtual memory spaces are independent each other. For example, users can create a virtual memory space that any process does not exist.

Resource Interface Controller: One of the main features of ***Tender*** is unified interface as I mentioned before. We call the unified interface "Resource Interface Controller (RIC)". Table 1 shows the unified interface on ***Tender***. There are five interfaces. "resource_name" is a pointer of strings, which indicate resource name. "pid" is current process id. "args" is a pointer of structure of arguments. "mod" is a mode of Resource Capability Control (RCC). "rid" is resource id.

These interfaces are called, when a program part of ***Tender*** operates a resource. RIC controls

Table 1. Unified interface of RIC.

open_rsc(resource_name, pid, args, mod)
close_rsc(rid, pid, args)
read_rsc(rid, pid, args)
write_rsc(rid, pid, args)
control_rsc(rid, pid, args)

the call of a program part of each resource. In order to control the call, RIC has a program pointer table, which is stored pointers of program parts of each resource. Figure 3 shows the program pointer table. This table is two-dimensional array. The rows of the program pointer table represent the type of operations (open, close, read, write, control), and the columns represent the type of resources. After RIC is called by any program parts of ***Tender***, RIC checks the resources name or resource id which is the first argument in order to get the type of resources. Then RIC searches a pointer which is indicated by type of resource and type of operations in the program pointer table, and calls a program which is addressed by the pointer.

Figure 5 shows overview of HVS implemented on ***Tender***.

Resource Capability Control: *Tender* has the access control mechanism in RIC. It is called "Resource Capability Control (RCC)"[4].

Every resources of ***Tender*** are assigned the mode of access right, when the resources are created. RCC has the access matrix. The rows of the access matrix represent subjects which are user or groups or others, and the columns represent resources. The access right consists of open, close, read, write and control.

The mode can be modified by the interface of RCC. RCC checks the capability of a target resource, before RIC calls a target program. If the access has the access right of the operation, RIC

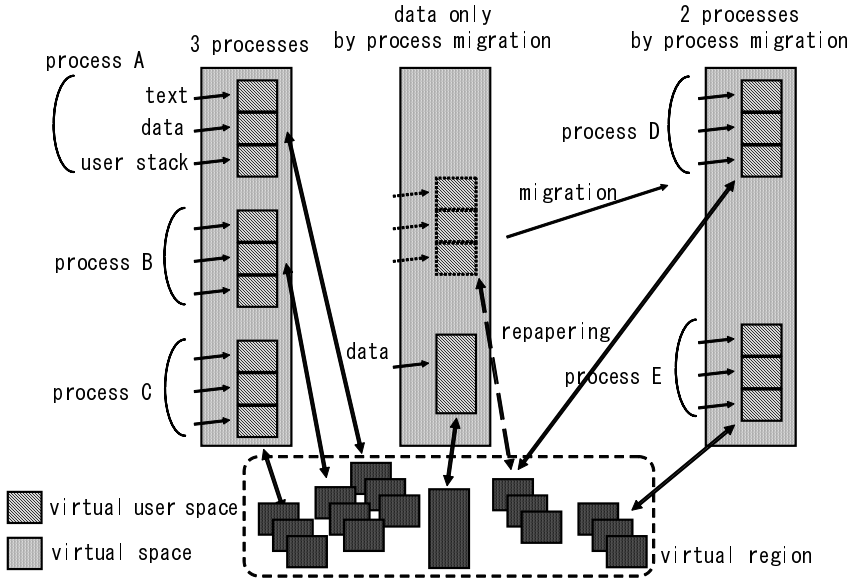

Figure 5. Resources used by processes.

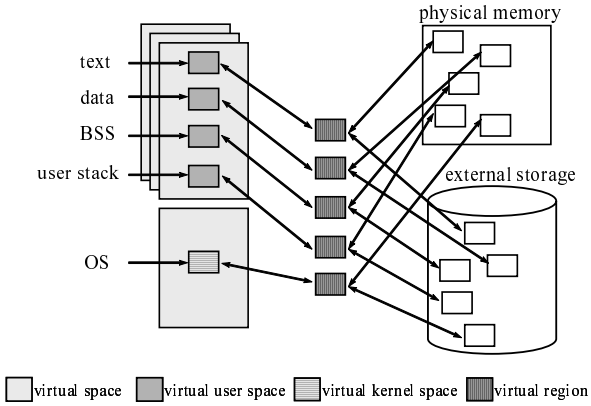

Figure 4. Memory resources on ***Tender***.

calls the target program. If not, RIC returns an error value to a caller program.

Memory Resources on *Tender*: Figure 4 shows memory resources on ***Tender***. In this figure, "virtual region" is a resource that virtualizes the data storage region information of the physical memory or the external storage. "Virtual space" is a space of the virtual address and corresponds to the mapping table where the virtual address is mapped into the physical address. "Virtual user space" is a space which is accessible from the processor by the virtual address. It is created by attaching "virtual region" to "virtual space" and deleted by detaching. Here, the attaching means to store the information of data storage region in the mapping table.

Problem of Implementation

In order to implement HVS, the problem that we have to consider is described as follows.

(1) Avoidance of stack address collision in process migration between virtual address spaces.

(2) Avoidance of address collision of program in process migration between virtual address spaces.

(3) Access control of process creation and migration

Allocation of Stack: HVS realizes more than zero processes can exist in a virtual address space, and a process can migrate other virtual address space. Thus, the stack address of each process has to be different among processes. We calculate the stack address by process id, and shift the stack address of each process. As a result, each process has a unique stack address.

Address of programs: The address of text region, data region and bss region of programs which access same files or communicate with each other has to be shift in order to avoid the collision of the address. Because not all programs need to run in same virtual address space, we recompile the programs and shift the address statically.

Access Control of Process Migration: We need security mechanism for HVS, because processes can migrate to other virtual memory space, and more than one process can exist on a virtual memory space. We realize that mechanism using capability. We assign capability which shows access right to each resource as described above. Subjects are owner, group and others. Objects are each resource. Types of operations are open,

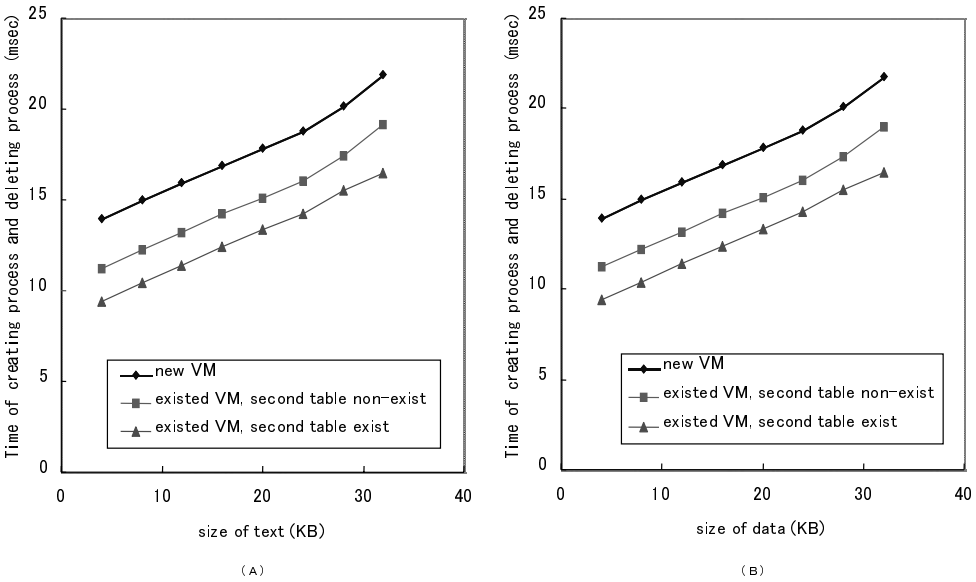

Figure 6. Evaluation result of process creation and deletion.

close, read, write and control. The access rights are check in resource interface controller. To create a process on existing virtual memory space, users need the right of write operations of the virtual address space. In order to make a process migrate to another virtual memory space, users also need the right of write operation of the virtual address space.

Advantages

There are three advantages of HVS.

(1) Processes can migrate to other virtual memory spaces.
On HVS, more than one process can exist on a virtual memory space. Thus, a process can migrate to other process's virtual memory space. As a result, processes on a same virtual memory space can share data and communicate with each other.

(2) Fast process creation and deletion.
Tender can create a process on an existing virtual memory space, because more than one process can exist on a same virtual memory space. It can reduce the processing time of process creation, because the process does not involve creation of virtual memory space. It can also reduce the processing time of process deletion.

(3) Reduction of overhead of inter-process communication.
If they communicate with each other frequently, we can reduce the overhead of context switch between them.

5 Evaluation

In order to evaluate the advantages of HVS, we measured the processing time of process creation and deletion. We also measured the processing time of process migration. We used a personal computer (Pentium 90MHz).

Virtual space consists of two steps of management tables. When virtual space is created, only the first table is created. The first table always exists, but the second table does not always exist, because the second table is created on demand in attaching virtual region. We can reduce the processing time of virtual space. For this reason, the processing time of process creation and process migration is affected by the existence of the second table.

The evaluation is performed on three conditions

(case 1) new virtual memory space (VM)

(case 2) existed VM, second table non-exist

(case 3) existed VM, second table exist

Process Creation

We made a benchmark program that create a process and delete it. The program repeats that processing 1000 times and outputs the average time of a process creation and deletion.

Figure 6 shows the results of the benchmark program on ***Tender***. We changed the size of text region (A) and the size of data region (B).

The processing time of process creation and deletion is proportional to the size of text region or data region. The processing time does not depend on type of region, because there is no difference of (A) and (B).

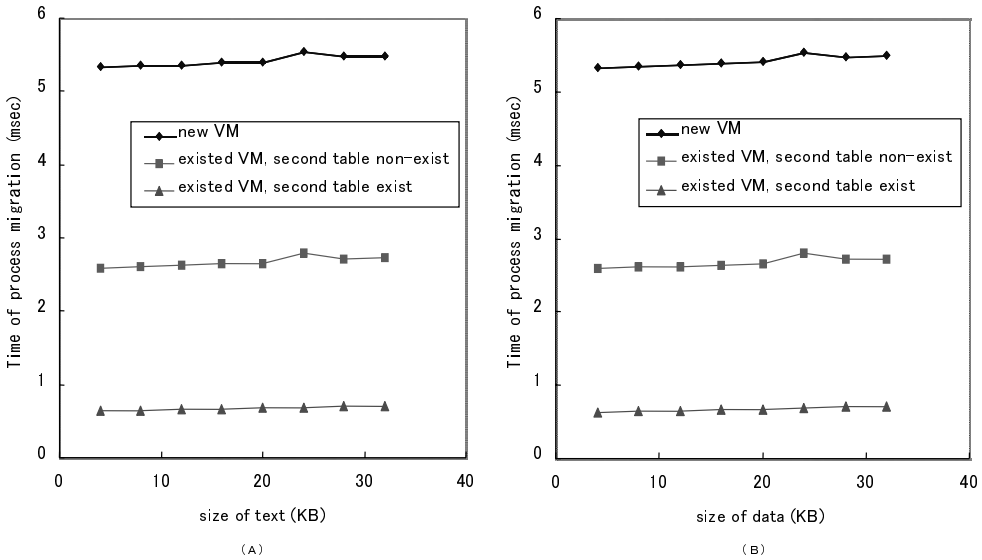

Figure 7. Evaluation result of process migration.

The processing time of case 2 is shorter than that of case 1 for 2.7 millisecond which is process time of virtual space creation. This time is creation of virtual space, the result shows the advantage 2 described above. The processing time of case 3 is shorter than that of case 2 for 1.8 millisecond which is processing time of creation of the second table of virtual space.

Process Migration

We made a benchmark program that migrate a process to other virtual space. The program repeats that processing 1000 times and outputs the average time of a process migration.

Figure 7 shows the results of the benchmark program on ***Tender***. We changed the size of text region (A) and the size of data region (B).

The processing time of process migration is proportional to the size of text region or data region. The processing time does not depend on type of region, because there is no difference of (A) and (B).

The processing time of case 2 is shorter than that of case 1 for 2.7 millisecond which is process time of virtual space creation. The processing time of case 3 is shorter than that of case 2 for 2.0 millisecond which is processing time of creation of the second table of virtual space.

6 Conclusion

In this paper, we propose Heterogeneous Virtual Storage (HVS). HVS provides multiple virtual address spaces. There are more than zero processes on a virtual space. A process can migrate between virtual address spaces.

We implemented HVS on ***Tender***, and explain ***Tender*** operating system and describe about the problems of the implementation, and the solution.

In order to evaluate the advantages of HVS, we measured the processing time of process creation and deletion, and process migration. That results show HVS can reduce the processing time of process creation and deletion. The processing time of process creation and deletion is reduced more than 1.8 millisecond. The processing time of process migration is reduced more than 2.0 millisecond.

As a future work, we will evaluate HVS by using application programs.

References

[1] 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, 2000, pp.3363–3374.

[2] S. C. Jeffrey , M. L. Henry, J. F. Michael, and D. J. Edward, "Sharing and Protection in a Single-Address-Space Operating System," **ACM Trans. on Computer Systems**, Vol.12, No.4, 1994, pp.271–307.

[3] L. Avraham, L. W. Joel, and S. Y. Philip, "Efficient LRU-based Bufferinf in a LAN Remote Caching Architecture," **IEEE Trans. on PDS**, Vol.7, No.2, 1996, pp.191–206.

[4] A. Yamamoto and H. Taniguchi, "A Method of Uniform Resource Capability Control on ***Tender***," **Proc. of 63th National Conventions of IPSJ**, Vol.1, 2001, pp.81–82.