LETTER *Special Section on Parallel and Distributed Computing and Networking*

Fast Control Method of Software-Managed TLB for Reducing Zero-Copy Communication Overhead

Toshihiro YAMAUCHI[†a)], ***Member***, **Masahiro TSURUYA**[†], ***Nonmember***, ***and*** **Hideo TANIGUCHI**[†], ***Member***

SUMMARY Microkernel operating systems (OSes) use zero-copy communication to reduce the overhead of copying transfer data, because the communication between OS servers occurs frequently in the case of microkernel OSes. However, when a memory management unit manages the translation lookaside buffer (TLB) using software, TLB misses tend to increase the overhead of interprocess communication (IPC) between OS servers running on a microkernel OS. Thus, improving the control method of a software-managed TLB is important for microkernel OSes. This paper proposes a fast control method of software-managed TLB that manages page attachment in the area used for IPC by using TLB entries, instead of page tables. Consequently, TLB misses can be avoided in the area, and the performance of IPC improves. Thus, taking the SH-4 processor as an example of a processor having a software-managed TLB, this paper describes the design and the implementation of the proposed method for ***AnT*** operating system, and reports the evaluation results of the proposed method.

key words: *microkernel, interprocess communication (IPC), software-managed TLB, operating system*

1. Introduction

Microkernel operating systems (OSes) [1], [2] architecture involves the implementation of near-minimum OS functions as a kernel; these include scheduling, memory management, and interprocess communication (IPC) functions. Other OS functions are implemented as processes (OS servers). OS servers include, among other functions, a file management function and various types of driver functions. Microkernel OSes can introduce new functions as OS server processes without modifying the kernel. OS servers are relatively reliable because most OS servers are developed by expert OS developers.

Microkernel OSes use zero-copy communication to reduce the overhead of copying transfer data, because the communication between OS servers occurs frequently in the case of microkernel OSes. However, zero-copy communication [3], [4], which occurs when sharing or replacing data between two virtual spaces, causes a translation lookaside buffer (TLB) miss during communication, because the associated page table must also be updated. In particular, embedded processors often have a software-managed TLB. When a memory management unit (MMU) manages TLB using software, TLB misses tend to increase the overhead, because the overhead of TLB misses in a software-managed TLB is larger than that of TLB misses in a hardware-managed TLB.

This paper proposes a fast control method for software-managed TLB that manages page attachment in the area used for IPC using TLB entries instead of page tables. In a previously proposed method [5], OS servers share the TLB entries of the data communication area with Application Program (AP) processes. In order to improve the IPC performance of OS servers, the proposed method allocates the TLB entries of the data communication area to OS servers. The proposed method can avoid TLB misses of OS servers in the area, and as a result, the performance of OS servers improves. This paper describes the design and the implementation of the proposed method for an operating system with Adaptability and Toughness (***AnT***) [6] that is based on the microkernel architecture. For this design and implementation, we consider the SH-4 processor that has a software-managed TLB. Furthermore, this paper reports the evaluation results of the packet transmission performance tests comparing the proposed method with two control methods of ***AnT*** and a method of ART-Linux [7], which is a monolithic kernel OS.

2. Control Method of Software-Managed TLB

2.1 Point of View

The zero-copy communication method has two overheads. First, the page table of the process needs to be updated while processing page attachment and detachment. Second, the TLB miss handler must be executed because the TLB entry related to page attachment and detachment must be flushed during its processing. In particular, when an MMU manages TLB using software, the TLB miss handler leads to a large overhead. Another problem is that the shared data can be destroyed by other processes that are part of the zero-copy communication of sharing a data between processes.

We took advantage of the software-managed TLB's ability to register information to the TLB by software. Specifically, we devised methods of registering information to TLB in order to avoid TLB misses in the data communication area, which is used to transfer data between processes. In this way, we are able to eliminate the need to update the page table. In addition, TLB misses do not occur when accessing the data communication area in IPC processing. Furthermore, TLB misses only occur when unattached area of data communication is accessed. The TLB misses are treated as data protection exceptions, and the number of

Manuscript received January 9, 2015.
Manuscript revised May 19, 2015.
Manuscript publicized September 15, 2015.

[†]The authors are with Graduate School of Natural Science and Technology, Okayama University, Okayama-shi, 700–8530 Japan.
a) E-mail: yamauchi@cs.okayama-u.ac.jp

DOI: 10.1587/transinf.2015PAL0003

TLB misses in the data communication area is decreased. Accordingly, the following are expected:

1. Update of the page table is unnecessary.
2. Reduction in processing overhead by avoiding TLB misses.
3. Protection of the data exchanged between processes by treating TLB misses as data protection exceptions.

2.2 Feature of SH-4 MMU

We describe the features of the SH-4 MMU in the following lines:

1. A multiple virtual storage (MVS) is divided into five areas that have different access credentials and address translation methods for each area; therefore, usage is limited. For example, the area to be used as a kernel space is directly mapped to physical memory and is accessed without checking the page table.
2. Users can choose from 1 KB, 4 KB, 64 KB, 1 MB, or a combination of different page sizes.
3. The TLB is composed of 64 entries. Users can choose any TLB entry with which to register the information of the page table. If users do not choose the TLB entry themselves, the system selects the entry based on the value of the random counter on the hardware.
4. If users want to use MVS, the TLB entry holds an identifier for each virtual space; therefore, it is not necessary to flush TLB entries when switching from one virtual space to another.

2.3 Fast Control Method of Software-Managed TLB

This paper proposes a fast control method of software-managed TLB. Our main goal is to manage page attachment in the data communication area by using TLB entries instead of page tables. When attached page exists in data communication area, the an entry of that page is always registered in TLB. Thus, TLB misses are avoided, and updating the page table becomes unnecessary.

With the proposed method, the control of the data communication area attachment is undertaken in the following manner:

1. When attaching the data communication area to a process of the AP, obtain a TLB entry and register the address translation information in the TLB entry.
2. When detaching the data communication area from the process, delete the address translation information and free the TLB entry.
3. OS servers are allowed to read from and write to the entire data communication area. The address translation information of the data communication area for OS servers is always registered in TLB entries.

In addition, the address translation information of the data communication area resides only in the TLB entry. Furthermore, this method does not use page table entries of the data communication area. If processes try to access the data communication area that is not attached, a TLB miss occurs. By treating TLB misses as data protection exceptions, this method can prevent data corruption and unauthorized access without including the overhead of checking the page table entry.

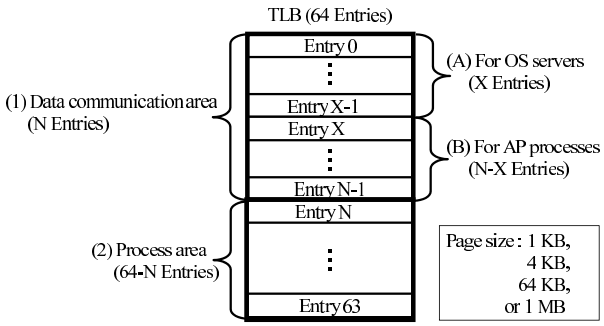

Fig. 1 Allocation of TLB entries.

This control method distinguishes AP processes from OS servers, and focuses on OS servers that are more reliable and communicate more frequently than AP processes. Thus, the TLB entry of the entire data communication area is always registered for each OS server. Therefore, it is not necessary to update page tables and TLB entries. In addition, the number of the TLB entries used by OS servers can be reduced by using a large page size in the data communication area.

In order for a control to make the best use of previously described features, it must allocate TLB entries in the data communication and other areas, such as the text part and the data part of the process area, and then manage those entries. Figure 1 shows the allocation of TLB entries.

The control method allocates N entries (from 0 to N – 1) to the data communication area. X entries are allocated to OS servers; therefore, the data communication area of AP processes can use up to $N - X$ pages. It allocates the remaining entries (from N to 63) to the process area.

In order to reduce TLB misses, the proposed method manages the state of the TLB entries that are allocated to the process area. Specifically, when they register the address translation information with the TLB, they choose the TLB entry that has not been used. If all entries have been used, the entry is selected based on the random counter on the hardware, thereby, reducing TLB misses better than using all the entries at random. In addition, we chose the page size and the number (N) of TLB entries to be allocated to the data communication area by considering the form of the IPC and operating environment.

There are three drawbacks of the proposed method. First, the reliability is reduced because OS servers can always access the entire data communication area. If an OS server destroys a memory region of the data communication area, it can negatively affect the system. Second, the number of OS servers that can run concurrently is restricted. The maximum possible number of concurrently running OS servers equals the number of TLB entries available for OS

servers. Third, the performance of TLB-intensive applications may degrade because number of TLB entries of process area is reduced.

3. Design and Implementation for *AnT*

3.1 *AnT* Operating System

The ***AnT*** operating system is based on the microkernel architecture. The program consists of the OS and service. The OS comprises of the kernel, which is referred to as the internal core, and the external core, which is executed as processes (OS server). Service consists of AP processes that execute the applications program. The internal core is the program component that guarantees the execution for a minimum number of common functions in all the systems. The external core is the necessary to adapt to the use cases of the systems. It has a dynamically reconfigurable structure. For example, ***AnT*** offers the functions of input/output control, file management, and device driver as processes. Service is the program component that offers services. The virtual space in ***AnT*** consists of multiple virtual storages.

In order to improve the IPC performance, ***AnT*** includes a data zero-copy communication function that uses an Inter-core Communication Area (ICA). This area is used by the internal core, the external core, and the service for data communication.

Inter-Server Program Communication (ISPC) mechanism [8] implements zero-copy communication between processes by using ICA. Specifically, the information about communication control and the arguments (request information) to be passed to the OS server are stored in the ICA for control (control ICA), and the transfer data is stored in the ICA for transfer data (data ICA). In addition, the kernel uses request and result queues for each process for communication. In addition, it offers the user a choice of synchronous or asynchronous communication.

3.2 Design and Implementation of the Proposed Method

AnT uses ICA as the data communication area. Therefore, we implemented ICA attachment and detachment. In this paper, we describe the implementations of the proposed method and the previously proposed method [5] for ***AnT***, which we will refer to as ***AnT*** proposed method and ***AnT*** previously proposed method, respectively. ***AnT*** previously proposed method is used for the evaluation of the proposed method. In addition, the original ***AnT*** considers the random counter on the hardware to choose the TLB entry to use as Linux for SH-4. In this case, the original ***AnT*** will be called ***AnT*** random. ***AnT*** random also searches all TLB entries and deletes the target entry, including the address translation information.

The implementation of ***AnT*** proposed method and ***AnT*** previously proposed method are as follows: The ICA attachment and detachment of AP processes register and unregister the address translation information to and from the TLB, instead of the page table. In addition, when unregistering the address translation information, it chooses the TLB entry directly, instead of using the random counter on the hardware. The OS servers of ***AnT*** proposed method need not perform these processes because OS servers do not share the TLB entries of the data communication area with AP processes.

4. Evaluation

4.1 Measurement Environment

For the test, we used a computer with an SH-4 (SH7751R 240 MHz) processor, and we used ART-Linux (kernel-2.4.29) for evaluations. ART-Linux also uses the random counter on the hardware to choose the TLB entry to use. In the case, ART-Linux will be called ART-Linux (random). In this study, we compiled ***AnT*** and ART-Linux using the same compiler (gcc-3.2.3). On ***AnT*** previously proposed method and ***AnT*** proposed method, we also allocated 32 entries of the TLB to the data communication area and the remaining 32 entries to the process area. A total of 8 TLB entries of the data communication area are allocated to OS servers on ***AnT*** proposed method.

4.2 Performance of Basic Operations

We describe the basic operations. When testing the basic operations, we measured communication time between the request AP process and the OS server. In this test, the control ICA does not have any arguments or return values to pass to the OS server. We evaluated synchronous request, asynchronous request, synchronous return, and asynchronous return. We also evaluated ISPC with data ICA and without data ICA. The data ICA size was 4 KB when data ICA was used. In order to determine the overhead of the communication mechanism, the OS server performs only the operations involved in the communication. The processing time of basic operations are shown in Fig. 2.

Each basic operation consists of process switch, first ICA access, ICA detachment, ICA attachment, system call, and other operations. Figure 2 shows that processing time of ***AnT*** proposed method is shorter than that of ***AnT*** previously proposed method by about 0.6–2.2 μs (about 6–19%). This is because the address translation information of the data communication area for OS servers in ***AnT*** proposed method always exists in TLB entries. It does not need to register and delete the address translation information in a TLB entry. In addition, Fig. 2 illustrates that the processing time of ***AnT*** proposed method decreased from ***AnT*** random by about 2.9–9.4 μs (about 29–49%). This results show the overhead of TLB misses of ***AnT*** random.

4.3 TCP/IP Communication Performance

AnT provides TCP/IP communication by using OS servers:

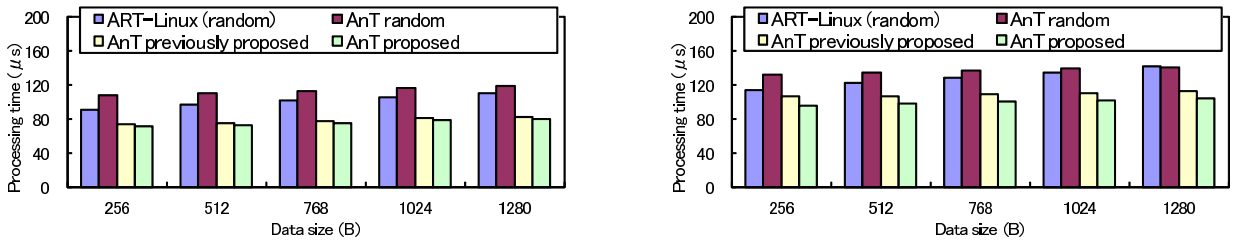

Fig. 3 Processing time of packet transmission (left: without other process, right: with other process).

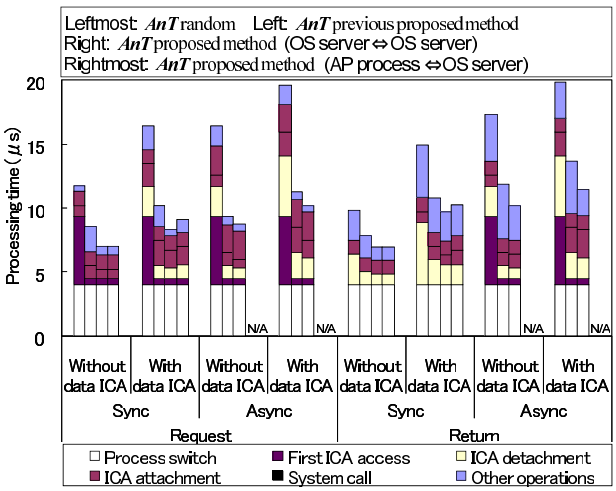

Fig. 2 Processing time of basic operations.

the network protocol control server and the Network Interface Controller (NIC) driver. We measured the processing time of one packet transmission by connecting the evaluation computer (Processor: SH7751R 240 MHz, NIC: Intel GD82551ER) and the receiver computer (Processor: Pentium4 3.4 GHz, NIC: Intel 82558, OS: FreeBSD 4.3-RELEASE) through a hub.

We evaluated two cases where another process exists and where another process does not exist. In the case, the coexisting process performs repeated memory accesses (read/write) to and from an area that is 32 KB (the same size as the data cache of SH7751R). In addition, the priority assigned to the coexisting process was the lowest among all processes. The processing time of packet transmission is shown in Fig. 3. The horizontal axis shows the size of a transmitted packet. From this figure, we can draw the following conclusions:

1. The packet transmission time of ***AnT*** proposed method was shorter than that of ***AnT*** previously proposed method by about 2.9–11.1 μs (about 4–10%). This is because the address translation information of the data communication area for OS servers in ***AnT*** proposed method always exists in TLB entries.
2. The packet transmission time of ***AnT*** proposed method was about 36.5–38.7 μs (about 26–34%) shorter than that of ***AnT*** random. This is because the overhead of ISPC was reduced.
3. The packet transmission time of ***AnT*** proposed method was about 18.0–37.8 μs (about 16–28%) shorter than that of ART-Linux (random). This is because ***AnT*** does not need to create the packet during the packet transmission operation. In contrast, ART-Linux creates memory region of packets during the packet transmission operation.

5. Related Works

In order to reduce TLB misses, the following methods have been proposed: utilizing TLB entries efficiently by enlarging the page size [9], pre-registering the information in the page table by estimating TLB misses [10], and scheduling processes to prevent TLB misses [11].

In [9], TLB misses are reduced by reducing the number of TLB entries per memory size, whereas in [10] registers the memory use in the TLB entry is registered before the memory is accessed. In contrast, the proposed method manages a specific area by using TLB entries without using page tables. The method proposed in [11] is valid only on threads running in the same virtual space and, therefore, cannot reduce TLB misses when communicating between processes, which is an issue addressed by the proposed method.

6. Conclusions

This paper proposed a fast control method for the software-managed TLB and described the design and implementation of the proposed method. The proposed method reserves the TLB entries of the data communication area of OS server. Consequently, the registration and deletion of the TLB entries are avoided in the area during IPC processing.

The evaluation results of the proposed method in ***AnT*** were shown. The ***AnT*** proposed method can improve the performance of IPC more than that of ***AnT*** previously proposed method by about 6–19% because the overhead of ISPC of OS servers is reduced. The ***AnT*** proposed method can also improve the performance of the packet transmission further compared with ***AnT*** previously proposed method by about 4–10%. These results show that the proposed method is useful because the communication of OS servers occurs frequently on microkernel OSes.

Acknowledgements

This work was partially supported by Grant-in-Aid for

Young Scientists (B) Number 25730046.

References

[1] J. Liedtke, "Toward real microkernels," Commun. ACM, vol.39, no.9, pp.70–77, 1996.
[2] A.S. Tanenbaum, J.N. Herder, and H. Bos, "Can we make operating systems reliable and secure?," Computer, vol.39, no.5, pp.44–51, 2006.
[3] D.L. Black, D.B. Golub, D.P. Julin, R.F. Rashid, R.P. Draves, R.W. Dean, A. Forin, J. Barrera, H. Tokuda, G. Malan, and D. Bohman, "Microkernel operating system architecture and mach," J. Inf. Process., vol.14, no.4, pp.442–453, 1992.
[4] K. Mouri and E. Okubo, "The design and implementation of the lavender micro kernel," Syst. Comput. Jpn., vol.31, no.7, pp.47–55, 2000.
[5] M. Tsuruya, T. Yamauchi, and H. Taniguchi, "Implementation and evaluation of software control method for TLB on microkernel OS," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J97-D, no.1, pp.216–225, Jan. 2014.
[6] ***AnT*** operating system, http://www.swlab.cs.okayama-u.ac.jp/lab/tani/research/AnT/index.html
[7] ART-Linux, http://www.dh.aist.go.jp/en/research/assist/ART-Linux/
[8] K. Okamoto and H. Taniguchi, "Implementation and evaluation of fast inter server program communication for ***AnT*** operating system," IEICE Trans. Inf. & Syst. (Japanese Edition), vol.J93-D, no.10, pp.1977–1989, Oct. 2010.
[9] T.H. Romer, W.H. Ohlrich, A.R. Karlin, and B.N. Bershad, "Reducing TLB and memory overhead using online superpage promotion," Proc. 22nd Annual International Symposium on Computer Architecture, vol.31. no.9, pp.176–187, 1995.
[10] A. Saulsbury, F. Dahlgren, and P. Stenstrom, "Recency-based TLB preloading," Proc. 27th Annual International Symposium on Computer Architecture, vol.28, no.2, pp.117–127, 2000.
[11] S. Yamada and S. Kusakabe, "Effect of context aware scheduler on TLB," IEEE International Parallel and Distributed Processing Symposium, pp.1–8, 2008.