Proc. Natl. Sci. Counc. ROC(A)
Vol. 22, No. 5, 1998. pp. 616-626

(Scientific Note)

Wide Address Translation and Protection for Single Address Space Operating Systems

SHIUH-PYNG SHIEH, ING-JYE SHYU, AND CHEN-YI SHEN

Department of Computer Science and Information Engineering
National Chiao-Tung University
Hsinchu, Taiwan, R.O.C.

(Received August 6, 1997; Accepted February 27, 1998)

ABSTRACT

In this paper, we present a new address translation and memory protection model to manage the wide 64-bit virtual address space, called the *segment-based translation and protection* (SBTP) model. It partitions a 64-bit virtual address space into 2^{32} segments with equal size of 2^{32} bytes. The SBTP model maintains a segment table to record used segments for each process. As a result of caching the per-process basis segment table on a designed memory cache, called the *segment look-aside buffer* (SLB), the virtual address translation time and protection rights verification time can be reduced. Furthermore, by separating the hardware mechanisms of address translation and protection, mapping information stored in the translation look-aside buffer (TLB) can be shared by all the processes and need not be flushed on each context switch. Thus, the cost of context switching compared with that conventional architectures is greatly reduced. Simulation results show that the proposed memory architecture effectively improves the performance of wide virtual address translation and memory protection for single address space operating systems.

Key Words: wide address, single address space operating systems, virtual memory simulator

I. Introduction

Because most current computer architectures are moving to 64-bit microprocessors, such as Alpha by DEC (Sites, 1992), PA-RISC by HP (Lee, 1989), R4400 by MIPS (Kane and Heinrich, 1992) and UltraSPARC by SUN (Yung, 1995), the single address space operating system concept has been proposed to utilize the wide (64-bit) virtual address space (Chase *et al.*, 1994; Bartoli *et al.*, 1993; Okamoto *et al.*, 1992; Chase, 1992). In these operating systems, all processes are spread over a 64-bit virtual address, where each process has its private portion in the 64-bit virtual address but can access others by issuing a 64-bit virtual address, as shown in Fig. 1. In this model, virtual address translation and memory protection are provided not through conventional address space boundaries, but through protection domains that describe the access rights of the pages or segments.

However, most 64-bit architectures are still housed 32-bit operating systems, such as UNIX, which is a 32-bit operating system and cannot fully explore the functionality of the 64-bit architecture (Koldinger, 1992), especially in virtual memory management. Therefore, in this paper, we address how the movement

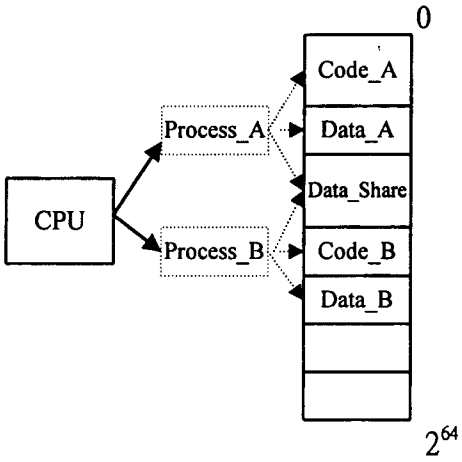

Fig. 1. Single address space environment.

to the 64-bit virtual address space and single address space operating systems affects traditional translation and protection methods on wide address architectures. We also propose a new scheme that can efficiently translate the wide 64-bit virtual address and effectively verify access rights for single address space operating systems.

The paper is organized as follows: In Section II,

we examine several virtual memory management architectures and address their drawbacks when they are applied to the single address space operating system with wide virtual address. In Section III, we present a virtual address translation and memory protection architecture, called the *segment-based translation and protection model* (SBTP model) which is specially designed for single address space operating systems with a 64-bit virtual address space. Section IV presents the simulation methodology and the simulation results by comparing different wide address architectures. Section V concludes this paper.

II. Related Virtual Address Management Architectures

In this section, we survey a number of memory management architectures and evaluate their applicability to wide virtual address translation and access right protection.

1. Virtual Address Translation Architecture

Virtual-to-physical address translation is an important factor determining system performance. In this section, we will present five translation architectures and explain why they are not suitable for translating wide virtual addresses.

A. Forward-Mapped Page Table and Guarded Page Table

The forward-mapped page table scheme uses the virtual address as an index to a hierarchical page table as shown in Fig. 2. The leaf nodes store page table entries while intermediate nodes store pointers to the next level. Considering the page table size, translating a 64-bit address space requires that the number of levels be up to seven or more. Thus, the overhead for resolving a translation look-aside buffer (TLB) miss is

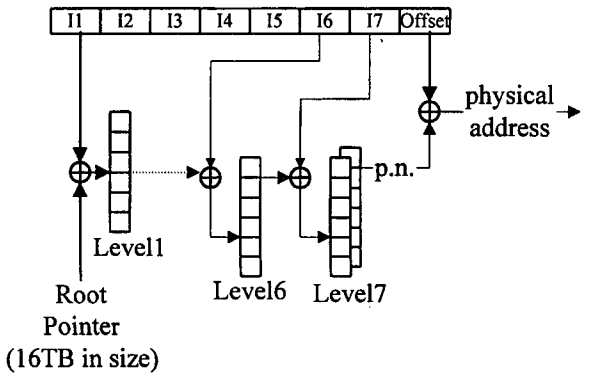

Fig. 2. Forward-mapped virtual address translation.

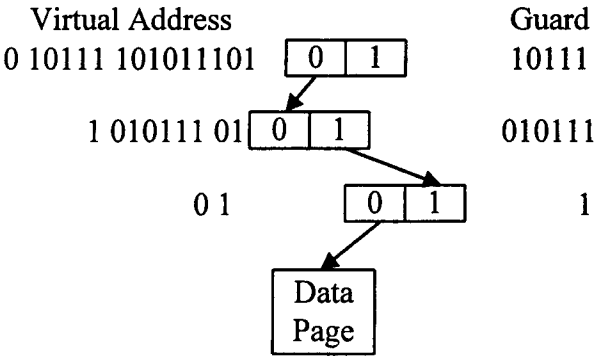

Fig. 3. Guarded page table.

seven memory accesses. In addition, the key drawback is that the implementation for the seven-layer page table will exponentially exhaust memory resources. For example, a forward-mapped page table of 16TB is required to cover a 2^{64}-bit virtual address space. Therefore, the scheme is not suitable for wide address translation.

The guarded page table scheme, a variant of the forward-mapped page table, has been proposed to handle sparse virtual address space management. As shown in Fig. 3, the top level of the page table is indexed using the upper bits of the virtual address. The guard is then compared with the remaining most significant bits of this address. If there is a match, the pointer contained in the entry is followed to the next level. The operation continues until all the bits are consumed. A guarded page table reduces both the translation time and memory overhead required for the tree-based page table. The drawbacks are that it is still a memory-traversed translation scheme, and that it does not provide protection and sharing information inherently.

B. Hashed Page Table and Clustered Page Table

Large address space systems often use hashed (inverted) page tables (Albert and Mergen, 1988; Huck and Hays, 1993). The simplest implementation uses an open hash table with a hash function that maps a virtual page number to a bucket. Each page table entry in the hash table stores the mapping information for one page, a tag identifying the virtual page number, and the next pointer. Hash tables use chaining to handle hash overflows (Fig. 4). During page table look-up, the hash function indexes into an array of hash nodes and traverses the hash bucket until a page table entry is found with a tag matching the faulty address. Extending hashed page tables to 64-bit addresses is straightforward, the difference is that the tag and pointers are now eight bytes each, resulting in sixteen bytes of overhead for each eight bytes of mapping information.

Clustered page tables (Talluri *et al.*, 1995), a

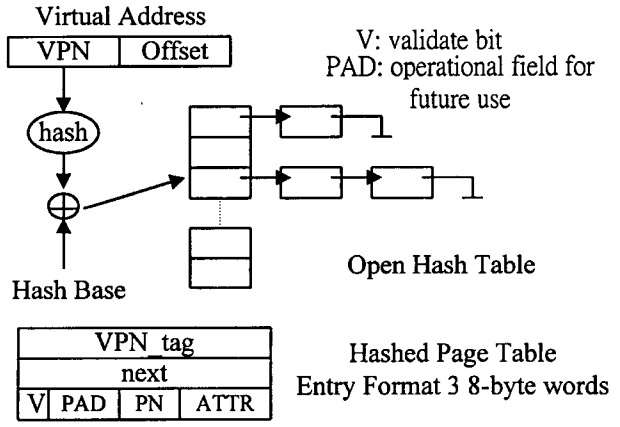

Fig. 4. Hash page table.

variant of hashed page tables, store mapping information for several consecutive pages with a single tag and the next pointer as shown in Fig. 5. Many page table operations in a clustered page table are similar to those in a hashed page table. During page table lookup, for example, the virtual page number is split into a virtual page block number (VPN) and a block offset (Boff). The VPN participates in the hash function, and the block offset indexes into the array of mappings in the page table entry (PTE) with a matching tag. Thus, if the virtual address space is sparse, the memory overhead is much less than that of linear page tables, and the translation time is much less than that of the original hash tables. In summary, hashed page tables and clustered page tables have less memory overhead, but their translation time highly depends on the hash function and the program locality.

C. Software Loaded TLB

The 64-bit architectures, such as MIPS processors (Manskey, 1991) or DEC Alpha (Sites, 1992), now support software loaded TLB, which turns page table design into an operating system issue (Uhlig, 1994). If a miss occurs, the processor traps to a fault handler, which is a part of the operating system. The handler searches a system-maintained table for address translation as well as protection information and then inserts it into the TLB. This approach hides the hardware complexity by means of software manipulation, but the latter requires an unpredictable amount of time for TLB miss resolution.

2. Protection Architecture

Another important issue in a multi-process operating system is that of providing efficient inter-process communication. However, bad design of objects sharing between processes will damage the involved processes. A well-designed protection architecture prevents a process from being damaged by other processes but allows it to share common memory resources with others.

A. Protection Look-aside Buffer Scheme

The protection look-aside buffer (PLB) scheme is a solution which provides parallel look-up for translation and protection information (Koldinger, 1992). In this scheme, the TLB contains the usual translation information, and the PLB contains protection information, such as access rights. In addition, process-specific validation data is cached on the PLB.

As shown in Fig. 6, the TLB and the PLB are searched in parallel. A successful TLB operation returns the physical address, and an unsuccessful one

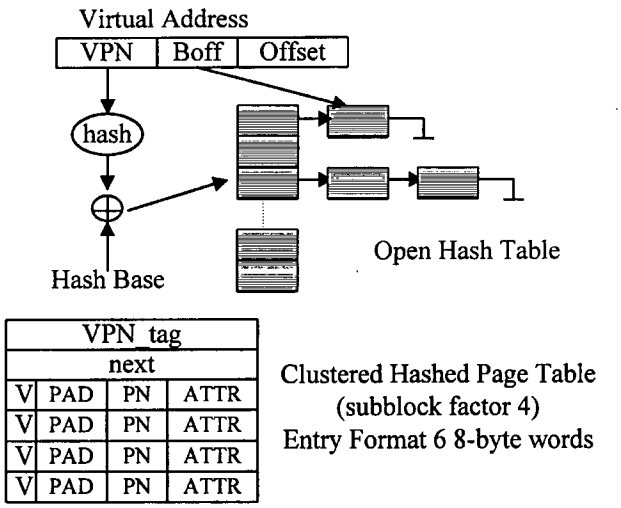

Fig. 5. Cluster hash table.

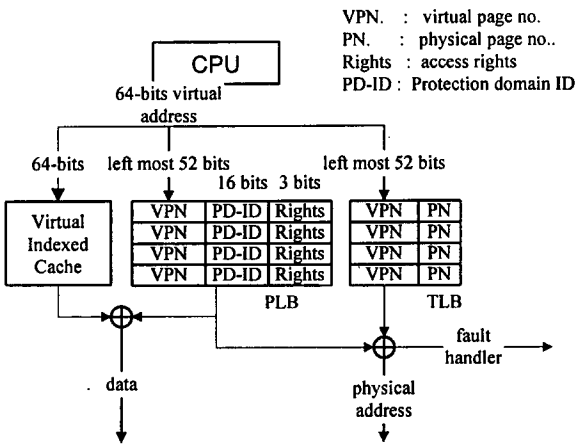

Fig. 6. Protection look-aside buffer.

generates a translation fault. The PLB search generates a protection fault if it is unsuccessful. The TLB is, thus, completely process-independent and does not need to be flushed at all during context switching. Furthermore, if the PLB entry is tagged with a process identifier, the PLB does not need to be flushed during context switching too. An improvement is to add a virtually indexed cache. It is noteworthy that the data cache can also be shared by different protection domains. The virtual indexed cache is a data cache indexed by a virtual address. In contrast, the physically indexed cache is a data cache indexed by a physical address.

B. Page Group Protection Model on HP PA-RISC

Segmented systems (e.g., the IBM 801 (Albert *et al.*, 1988), INTEL X86 and HP PA-RISC (Lee, 1989)) support uniform sharing to some degree. The first phase of address translation on segmented architectures concatenates a global segment identifier with a segment offset, yielding a long-form address from a global virtual address space. The segment identifier is retrieved from a segment table or a vector of segment registers associated with the current domain. The segment table is typically accessed by the high order bits of the domain specific address. Domains define a local view of portions of the global address space by overlaying global segments into their private segment registers. Segmented systems are able to implement a high degree of sharing and module protection, but they are not suitable for implementation of single address space operating systems because, except for the HP PA-RISC architecture, they lack a cross-segment pointer. A page-group model implemented in HP PA-RISC defines logical groupings of pages. Each page is a member of a single page-group, and a protection domain is defined by the set of page-groups that it can access. In PA-RISC, the TLB entry for a page includes a set of access rights and a field called an access identifier (AID) that contains a page-group number, in addition to translation information, as shown in Fig. 7.

In each memory access, the TLB is searched using the virtual page number, and returns the physical address translation and the AID for the page. The processor must then determine if an access to the page-group specified in the AID is permitted in the current protection domain. If the page-group number in the AID matches one of the PIDs, or if the AID field is zero, then the exact access rights allowed are determined by a combination of: (1) the access rights specified for the page in the Rights field of the TLB entry, (2) the current processor privilege level (PL in Fig. 7), and (3) a write-disable bit in the PID register (D in Fig. 7). The set of page-groups accessible to the current domain is stored in a set of four page group registers. Because the page-group model requires only one set of access rights per page, protection and translation information can be combined in a TLB without duplicating translation data. However, four page-group registers do not satisfy the needs of single address space operating systems, which always attach 4~16 segments to each protection domain. Thus, a fault handler is required which replaces the protection information, and the performance of the model is determined by the efficiency of the handler. In the original designs of both the PLB scheme and the Page Group Protection Model, there was no hardware facility to speedup the retrieval of protection information when a cache miss occurs.

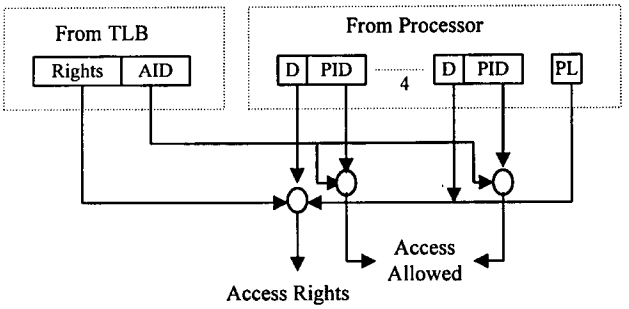

Fig. 7. Page-group protection model on HP PA-RISC.

III. Segment-Based Translation and Protection

In this section, a scheme for virtual address translation and protection, called the segment based translation and protection model, will be proposed to improve the performance in 64-bit virtual address translation and protection. The SBTP model can also be applied to virtual memory architectures that use software loaded TLB by slightly modifying the TLB fault handler. Thus, the cost of flushing the TLB for each context switch can be greatly reduced.

1. SBTP Model

A translation and protection model in a single address space operating system defines the range of a 64-bit virtual address space to which each process can access. In our proposed SBTP model, the left-most 32 bits of a 64-bit virtual address represents the segment identifier, and the other 32 bits represent the offset in a segment, as shown in Fig. 8. The segment is a minimum protection and allocation unit provided by

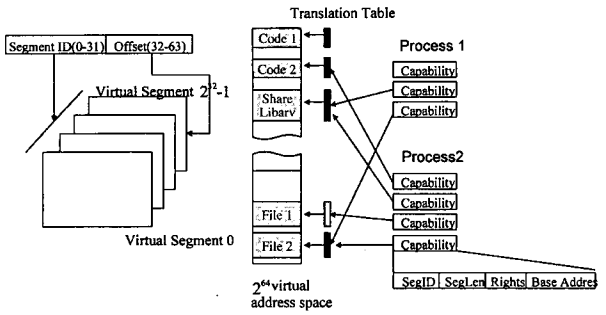

Fig. 8. Single address space on a segment-based protection model.

the virtual memory management. Each segment plays the role of a stack, temporary heap, memory mapped file or library code.

With the SBTP model, the operating system kernel must maintain translation tables for each segment and protection tables for each process. Primitively, each process running on such a system can access (read/write/execute) any data or code segment in the whole address space (2^{32} segments) if this process has the access capabilities of all the segments. However, a process is limited to access of some segments to which the process has access capability. A collection of capabilities of a process forms a *protection domain* of this process. Each capability consists of a 32-bit segment identifier, a 5-bit long segment, a 4-bit access right owned by the process and a 20-bit base address pointing to a translation page table for the segment. Thus, in total, eight bytes are needed for a protection table entry. The 20 bits from 32 to 51 in the 64-bit virtual address are used as an index to a two-level page table for translation. The 5-bit long segment is sufficient to record the usage of the 20 bits ($2^5 = 32 > 20$). When the value of the segment length field is smaller than 10 (less than 10 bits are used), the SBTP model speeds up by looking up the second-level page table instead of the first-level page table.

2. Hardware-supported SBTP Model

To reduce the time needed for virtual address translation and protection verification, a new memory cache called the *segment look-aside buffer* (SLB) is introduced. SLB is a fully associative cache with 16~32 entries. It is used to cache the protection table entries used by the SBTP model. In context switching, the SLB is flushed and then filled with the protection table entries of the new process. Thus, the SLB contains the capabilities of the new process, and SLB cache faults never occur within the time slice unless illegal references to an unattached segment occur.

A virtually indexed cache is also embedded into the memory architecture. Figure 10 depicts the high-level organization of the SLB, TLB and a virtually indexed cache. At each memory reference, SLB, TLB and the virtually indexed cache are accessed in parallel. The segment identifiers are looked up in the SLB, which is extracted from the left-most 32 bits of the virtual address. The virtual page number is looked up in the TLB, which is extracted from the left-most 52 bits of the virtual address. The whole 64-bit virtual address is looked up in the virtually indexed cache for a data hit or miss. The lock set bit (l.s.) in the TLB provides a page-level lock mechanism. The owner of a segment can set its page's lock bit in the TLB to prevent other processes from accessing this page. This page-level lock functionality is widely demanded by database applications.

During operations on these three cache structures, several conditions can occur:

(1) Condition 1: If cache hits occur on the SLB, TLB and the virtually indexed cache, the protection information and the lock information are extracted directly from the entries of the SLB and TLB, and access rights are verified at the same time. If it is a legal reference, a data block in the virtually indexed cache is accessed immediately.

(2) Condition 2: If cache hits occur on the SLB and TLB, but a miss occurs on the virtually indexed cache, virtual-to-physical address translation is performed. Translation mapping information is then stored in the TLB. At the same time, the access right is verified, and the instruction is restarted.

(3) Condition 3: If a hit occurs on the SLB only, the access right is verified, and the base address of the page table of the address's segment is extracted by the TLB fault handler, which searches for this segment in the page table. A two-level forward-mapped page table is used to handle the remaining 32-bit virtual address translation, and each level is indexed with a 10-bit virtual ad-

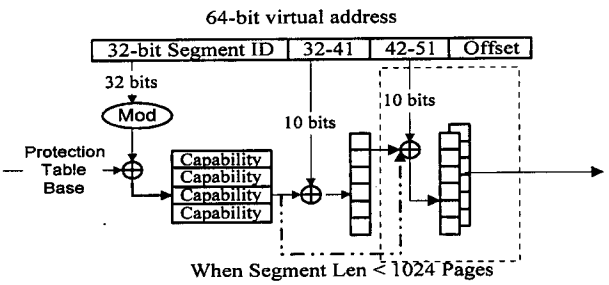

Fig. 9. Segment-based protection model.

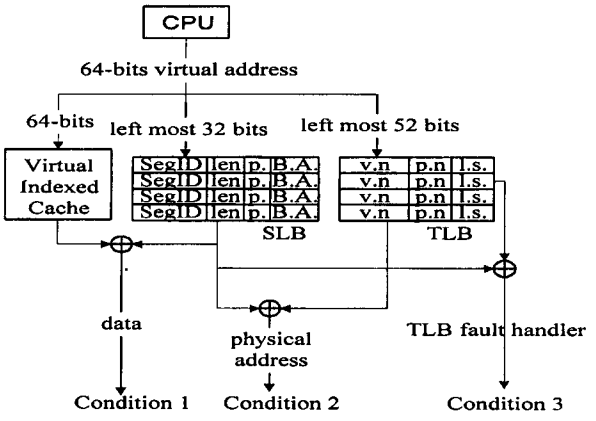

Fig. 10. Segment look-aside buffer.

dress, as also shown in Fig. 9. If the segment size is less than 1024 pages, only a one-level page table needs to be traversed. After the TLB mapping information is refilled by the fault handler, the instruction is restarted.

(4) Condition 4 : If no match occurs on the SLB or if the access is illegal, the protection fault handler is signaled to refill the SLB and the protection table, or it sends an illegal reference signal to the current protection domain.

The SBTP Model combining the SLB, TLB and virtual indexed cache has the following merits:

(1) By separating the hardware support mechanisms for translation and protection, translation information can be shared by all the processes and does not need to be flushed in every context switching. Thus, the cost of flushing the TLB for context switching is greatly reduced.

(2) By caching higher order virtual address translation information in the SLB, the TLB refilling time is reduced. Only one or two level page tables need to be traversed when a miss occurs in the TLB.

(3) The number of cache misses on the SLB is much smaller than that on a page based protection cache such as the PLB scheme. Furthermore, in our SBTP model, the protection table can be represented as a hashed table. Thus, the protection cache refilling time can be greatly reduced.

(4) By maintaining page based protection on the TLB, some page operations can be supported in our segment based protection model.

3. SBTP Model with Modified Software-Loaded TLB Support

In a software-loaded TLB architecture, the TLB entries are intended for individual address spaces which combine translation and protection information, and need to be flushed every context switching. The flushing operation removes not only the protection information, which is different for each process, but also the translation information, which is identical for all processes. Improvements for the flushing overhead are provision of validate/invalidate bits in the TLB entries and separate translation and protection tables in the TLB miss handler. On each protection domain switching (context switch), the system invalidates all the TLB entries (sets the validate bit FALSE but does not flush it), and when a TLB miss occurs (caused by the invalidate state), the TLB miss handler searches the protection table for a match to the virtual page number if there is an entry indexed by the faulted virtual page number in the TLB. If access to the address is permitted, the handler only needs to validate the TLB entry to complete the cache miss. Otherwise, the TLB fault handler searches both the protection and translation tables to reload the TLB entry. If we can guarantee that the searching time in the protection table is less than that in the translation table, the cost of context switching may be reduced by invalidating the TLB entries instead of flushing. Therefore, we modify the software-loaded TLB structure to provide fast protection table lookup in the SBTP model.

Shown in Fig. 11 is a hybrid scheme which uses both the forward-mapped page table and the hashed page table. If a virtual page number matches the invalidated TLB entry, only the hashed table needs to be traversed to find a corresponding protection table entry in the per-process basis protection table. If a translation information fault occurs, the hash page table is traversed first to find the protection information and the base address of the page table, and a one or two-level forward-mapped page table is traversed to find the translation information. The simulation in Section IV shows that the translation time and memory overhead of this modified software-loaded TLB method are reduced.

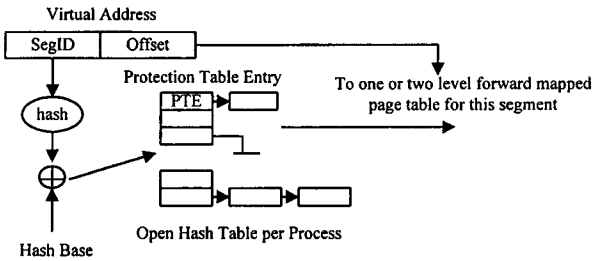

Fig. 11. Applying the SBTP model to the modified software loaded TLB.

IV. Simulation and Performance Evaluation

In this section, we will present the simulation methodology, give the simulation results based on comparison of different wide address architectures and show that our approach performs better than others.

1. Simulation Model

Since a 64-bit operating system environment was not available, we built an environment which could effectively simulate the 64-bit virtual memory access pattern of the processes in a system. We implemented a 64-bit virtual address simulation environment on a 32-bit operating system. The simulation environment, called the Single Address Space Operating System (SASOS) simulator, was built upon Linux Slackware 1.2.3, which collects the 32-bit memory reference behavior of the simulated processes and then generates 64-bit address traces as input to the hardware simulator.

Figure 12 shows the relationship between the 32-bit workloads, the SASOS simulator, the memory architecture simulator, the hardware simulator and the Linux operating system. The SASOS simulator was implemented through the UNIX system call ptrace(), which allows us to trace a process in single step mode and thus the memory reference and register values can be easily intercepted. The workload returns execution right after every instruction execution, and the SASOS simulator can grab the workload's memory reference and produce 64-bit virtual address which is then sent to the hardware simulator for advance simulation. The SASOS simulator maps 32-bit memory reference behavior to the 64-bit single address space. A 64-bit virtual address is generated by combining the 32-bit

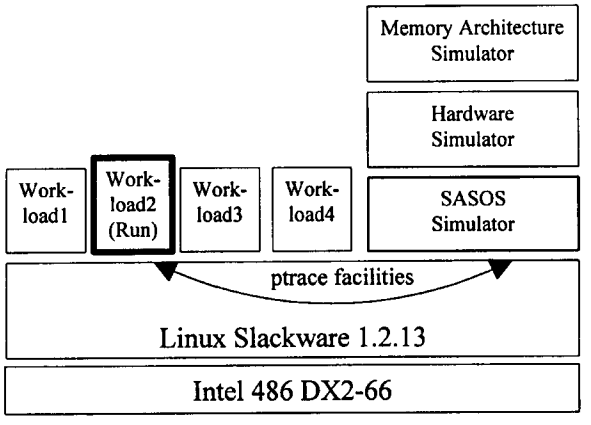

Fig. 12. Relationship between different simulators of our simulation.

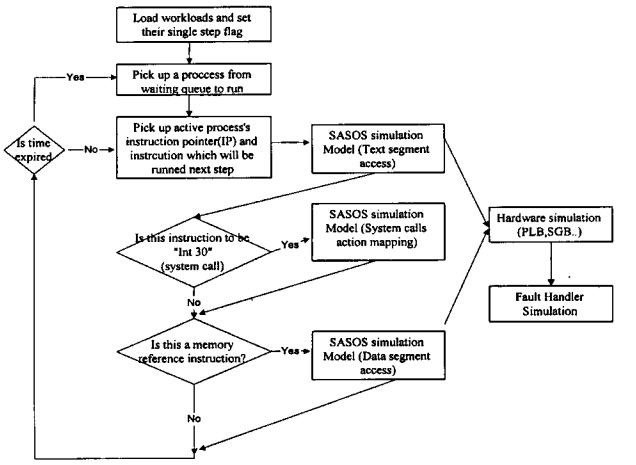

Fig. 13. Simulation process.

segment ID and its 32-bit offset. The reason why we used 32-bit benchmarks as input forour simulation is that a 64-bit environment and 64-bit program where not available. We needed an approach which could produce the program trace without any loss of correctness, thus, we implemented the simulation environment to on-time simulate 64-bit memory access behavior under 32-bit operating systems. There were three major access patterns in the simulator: text segment accesses, data segment accesses and system calls. Figure 13 shows that we intercepted each mmap() or mprotect() system call to create a new segment for use in the SBTP model and re-mapped the data or code access address to a new 64-bit virtual segment address. We also implemented various hardware architectures in the hardware simulator module, including the conventional TLB scheme, the modified TLB scheme, the PLB scheme, the modified PLB scheme and the proposed SLB scheme. When a simulated hardware cache fault occurred, the memory architecture simulator was invoked. The memory architecture simulator also built several schemes, including forward mapped, hashed and clustered page tables schemes.

2. Evaluation Consideration

We evaluated these schemes using metrics of memory-overhead, which consider the memory space consumed in maintaining virtual address mapping and protection domain specification as well as translation and protection time, which is the time spent searching the memory hierarchy for translation and protection information. The address translation and protection time can be calculated as follows :

$$T_{translation} = R_{hit} \times T_{hit} + R_{miss} \times T_{miss},$$

Table 1. Workload Characteristics

	Process number	Segment number	Number of shared segments	Number of segments per process
Arithmetic(1)	6	21	0	7
Compress(6)	6	32	2	7
Dhrystone(6)	6	32	2	7
Gcc(2)	2	11	1	5
Misc(5)	5	27	2	7
System(2)	6	36	6	6

where

$T_{translation}$:Translation and Protection time needed to translate a 64-bit virtual address to a physical address;

T_{hit} : translation time for a TLB hit;

T_{miss} : translation time for a TLB miss;

R_{hit} : probability of a hit;

R_{miss} : probability of a miss.

If a TLB hit occurred, the translation time was the same no matter which approaches were used. Therefore, we only compared the time spent handling a TLB miss. Our simulation environment did not allow us to measure such case directly. Instead, we used the average number of memory access to handle a TLB miss as an indirect metric (Talluri *et al.*, 1995). This metric would be proportional to the page table access time if the (second level) cache rarely contained page table data and other overheads was minimal.

Different protection architectures affect the miss ratio of the TLB, and the miss ratio of the TLB also impacts the system performance very much. We compare three architectures below:

(1) The conventional memory architectures: TLB is used for a single process and need to be flushed at each context switching. TLB faults occur frequently after context switching.

(2) The TLB is tagged with the process identifier: the TLB does not to be flushed when a context switching occurs. However, different processes occupy different TLB entries if they share the same virtual page.

(3) The separated translation and protection scheme TLB is shared by all the processes but not the PLB.

Finally, we studied the miss ratio of the protection look-aside buffer on different memory architectures. The miss ratio in the protection cache heavily affects the system performance.

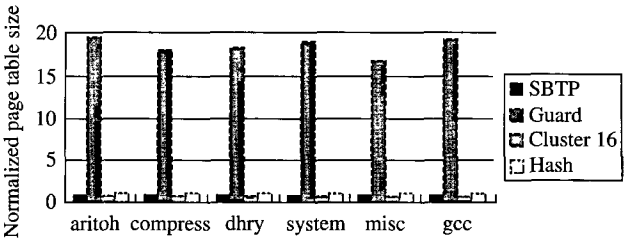

Fig. 14. Memory overhead of translation schemes with a single translation page table.

3. Simulation Results

Six generic programs where selected as workloads, including: **Gcc** and **Compress**, which come from the SPEC92 (Reilly, 1991) suite and are generic multiprogramming benchmarks; **Arithmatic**, **System**, **Misc** and **Dhrystone**, which come from the Byte (Price, 1989) benchmark suite and are commonly used as UNIX system performance benchmarks. Each of them contains several programs used to evaluate different system performance factors. We invoked multiple copies of these benchmarks at each instant. Table 1 lists related parameters. The first column shows the workload name, and the number in parentheses specifies the number of copies run at the same time. The second column shows the maximum process number when the workload is running. The third column represents the total number of segments allocated on this workload. Also, the number of shared segments and the number of segments per process are shown in the last two columns individually.

The first simulation was conducted to determine the impact of the page table size. Figure 14 shows relative page table sizes, which are normalized to hashed page table for various workloads when the system

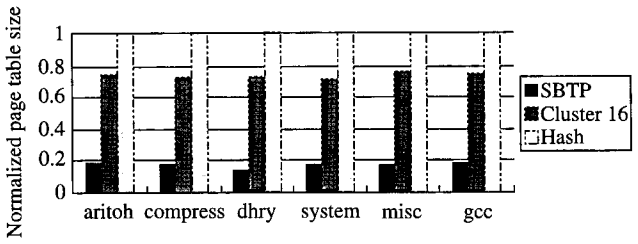

Fig. 15. Memory overhead for separating translation and the protection table.

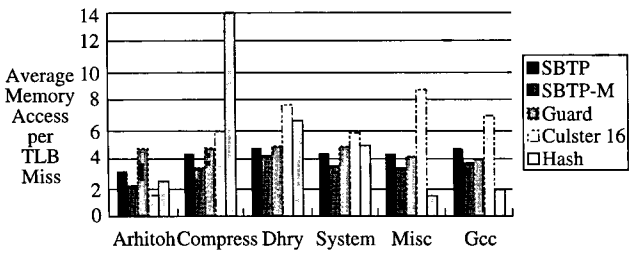

Fig. 16. Translation time for different translation schemes.

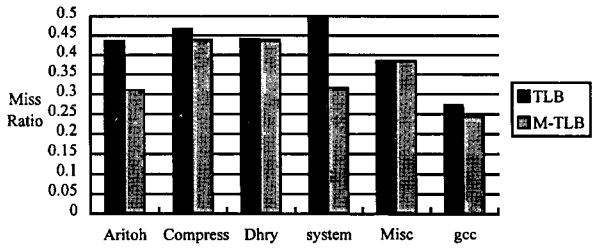

Fig. 17. Miss ratio on the TLB and modified TLB with 128 entries.

maintains a single translation table and multiple protection tables. The simulation result shows that the clustered page table with 16 sub-blocks needs less memory overhead, and that the SBTP model can reduce the memory overhead to a reasonable size which is lower than that of the hashed page table in general. Though the guarded page table greatly reduces the memory overhead compared to the original forward mapped page table, it is still not applicable to 64-bit virtual address architectures, unless the page table is stored in kernel address space (Silha, 1993).

Figure 15 displays relative page table sizes which are normalized to hashed page table sizes for various workloads when the system maintains multiple page tables for all processes. The simulation result shows that the SBTP model has the least memory overhead, because the segment translation page table is shared.

The second simulation compared the translation time needed by different page table structures. In Fig. 16, the simulation results show that the SBTP needs fewer memory accesses when the TLB is refilled. In the clustered and hashed page table schemes, the number of memory accesses per TLB fault depends on the workload of the system, because different workloads have different memory access patterns. Figure 16 also shows that the SBTP model with a modified software loaded TLB reduces the cost of flushing the TLB at every context switching by reserving shared TLB entries. The translation time for the shared entry is reduced to the time needed to traverse the protection table only.

Figure 17 shows a comparison of the TLB miss ratios on different architectures. With the hardware separating protection and translation mechanism, when the TLB (S-TLB) is shared by all the processes, the miss ratio on the shared TLB architecture is reduced. Compared with a TLB tagged with a process identifier (T-TLB), the S-TLB reduces the use of the TLB entries which are mapped to the same translation information

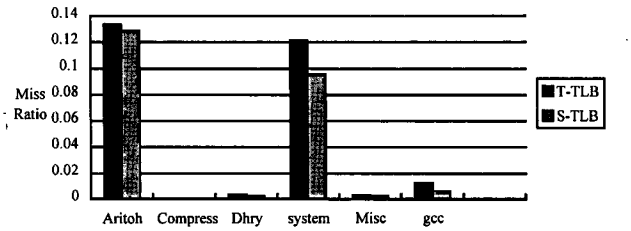

Fig. 18. Miss ratio on a TLB tagged with a process identifier.

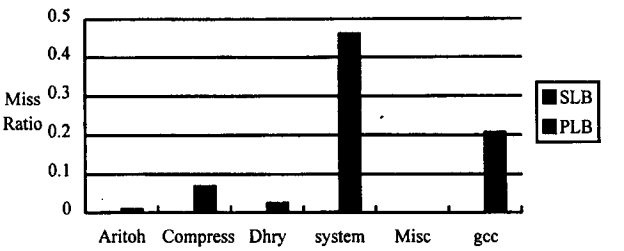

Fig. 19. PLB and SLB miss ratio per context switch (32 entry fully associative PLB and SLB).

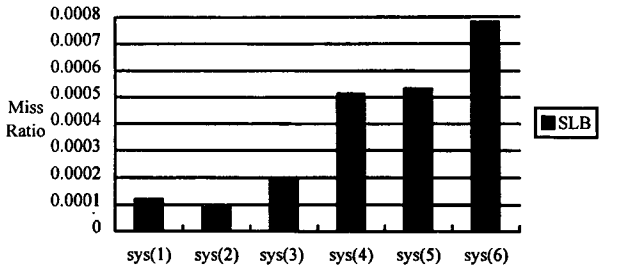

Fig. 20. SLB miss ratio per context switch with 16 cache entries.

by different processes, as shown in Fig. 18.

The miss ratio of the PLB and SLB tagged with the process identifier is shown in Fig. 19. It is predictable that the segment based look-aside buffer (SLB) has lower miss ratio than page based look-aside buffer (PLB). The result reveals two additional pieces of information. First, the PLB miss ratio is high for some workloads. Because of the lack of an efficient protection data structure, the performance of the PLB system may be poor for some workloads. Second, on average, the SLB miss ratio is almost zero. The SLB with the tagged process identifier has better performance than the SLB which purges protection data at each context switching. Figure 20 shows that SLB misses seldom occur, even when the size of the SLB is reduced to 16 entries. This is because of the memory reference locality; the small SLB (16~32 entries) with the tagged process identifier is sufficient for caching segment-based protection information.

V. Conclusion

In this paper, we have proposed a virtual memory protection and translation model, called the SBTP model, and a segment look-aside buffer to support memory management for a single address space. We also have presented a modified software-loaded TLB scheme for this model if a system does not support a hardwired segment look-aside buffer. Simulation results show that the proposed model effectively improves the performance of virtual address translation and protection verification for single address space operating systems on wide address architectures.

Acknowledgment

This work was supported by the National Science Council, R.O.C., under research project NSC 85-2221-E009-039.

References

Alert, C. and M. F. Mergen (1988) 801 storage: architecture and programming. *ACM Trans. on Computer Systems*, **6**(1), 25-50.

Bartoli, A., S. J. Mullender, and M. V. D. Valk (1993) Wide-address space-exploring the design space. *Operating Systems Review*, **27**, 11-17.

Chase, J. S. (1992) *How to Use a 64-Bit Virtual Address Space*. Technical Report 92-03-0, Dept. of Computer Science and Engineering, Univ. of Washington, Seattle, WA, U.S.A.

Chase, J. S., H. M. Levy, M. J. Feeley, and E. D. Lazowska (1994) Sharing and protection in a single-address-space operating system. *ACM Trans. on Computer Systems*, **12**(4), 271-307.

Huck, J. and J. Hays (1993) Architectural support for translation table management in large address space machines. *20th Annual International Symposium on Computer Architecture ISCA '20*, pp. 39-50. San Diego, CA, U.S.A.

Kane, G. and J. Heinrich (1992) *MIPS RISC Architecture*. Prentice Hall, Englewood Cliffs, NJ, U.S.A.

Koldinger, E. J. (1992) Architectural support for single address space operating systems. *Proc. of the 4th Conference on Architectural Support for Programming Language and Operating System*, pp.175-186. Santa Clara, CA, U.S.A.

Lee, R. B. (1989) Precision architecture. *IEEE Computer*, **22**(1), 78-91.

Manskey, J. R. (1991) 64-bit computing. *Byte*, **9**, 135-142.

Okamoto, T., H. Segawa, S. H. Shin, H. Nozue, K. I. Maeda, and M. Saito (1992) A micro kernel architecture for next generation processors. *Proc. of USENIX 1992 Winter Conference*, pp. 83-94. Monterey, CA, U.S.A.

Price, W. (1989) A benchmark tutorial. *IEEE Micro*, **89**, 28-43.

Reilly, J. (1991) SPEC. *SPEC Newsletter*, **3**(4), 2-8.

Silha, E. (1993) *The PowerPC Architecture*. IBM RISC System/6000 Technology, IBM Corp., White Plains, NY, U.S.A.

Sites, R. L. (1992) *Alpha Architecture Reference Manual*. Digital Equipment Corp., Maynard, MA, U.S.A.

Talluri, M., M. D. Hill, and Y. A. Khalidi (1995) A new page table for 64-bit address spaces. *ACM Operating System Review*, **29**(5), 184-200.

Uhlig, R., D. Nagle, T. Stanley, T. Mudge, S. Sechrest, and R. Brown (1994) Design tradeoffs for software-managed TLBs. *ACM Trans. on Computer Systems*, **12**(3), 175-205.

Yung, R. (1995) *UltraSPARC-I Architecture*. Technical Report, Sun Microsystems, Palo Alto, CA, U.S.A.

單一位址空間作業系統下高位元虛擬位址轉換與保護

謝續平　徐英傑　沈長毅

國立交通大學資訊工程研究所

摘　要

傳統的虛擬記憶體管理多被用於像UNIX一樣的私有虛擬空間的作業系統，但並不能有效地應用於單一位址空間作業系統。在這篇論文中，我們提出一個新的虛擬記憶體保護模組來管理單一位址空間。藉著將64位元的虛擬位址空間分成2^{32}個2^{32}位元的區段，作業系統必須管理關於每個行程的區段表以及每個區段的位址轉換表。每個區段表上記錄這個行程可存取的區段以及區段的位址轉換表的位置，藉著將區段表暫存在我們所提出的一個新快取記憶體架構上，位址之轉換及存取權限之檢查的時間都會縮短。而且藉由分開的位址轉換及位址保護快取架構，傳統耗費在行程轉換所作的虛工也大量被減少。在我們的模擬結果中顯示，這個虛擬記憶體管理架構的確增加了在單一位址空間作業系統中虛擬記憶體轉換及保護的效能。