UDC 004.451.3

Y. I. KLIMIANKOU

TRANSLATION LOOKASIDE BUFFER MANAGEMENT

Belarusian State University of Informatics and Radioelectronics

This paper focuses on the Translation Lookaside Buffer (TLB) management as part of memory management. TLB is an associative cache of the advanced processors, which reduces the overhead of the virtual to physical address translations. We consider challenges related to the design of the TLB management subsystem of the OS kernel on the example of the IA-32 platform and propose a simple model of complete and consistent policy of TLB management. This model can be used as a foundation for memory management subsystems design and verification.

Keywords: *virtual memory, physical memory, memory management, TLB management.*

Introduction

The virtual memory was made to cope with growing complexity of computer software. This technique allows operating system to split software package executing on computer into isolated parts called processes and at the same time isolate OS kernel from application software. Virtual memory is supported by most of the widespread advanced processor architectures such as IA-32 [1], ARM [2] and MIPS [3].

Memory Management Unit (MMU) handles accesses of processor to main memory. MMU performs translation of virtual memory addresses into physical memory addresses [4]. Such translation introduces significant performance penalty. Translation Lookaside Buffer (TLB) is a special MMU cache which was introduced to reduce that performance loss.

This paper is focused on the TLB management as part of the memory management performed by operating system kernel. Our goal is a consistent, efficient and self-sufficient TLB management model atop of which complete memory management subsystem can be built. Developed model was designed with primary focus on its application in second generation microkernels.

We consider TLB management as a bottom layer of memory management [5]. Actual policy and/or mechanism of memory allocation is expected to be implemented on the basis of TLB management layer. However, memory management in general is a different topic and is not considered in this paper.

Gorman in [6] presented the overview of the TLB management and its place in memory management on the example of Linux kernel. However, this overview was done in the context of monolithic kernels where TLB management functionality tends to be smeared between modules. In contrast, we propose concise model suitable for true microkernels.

Translation Lookaside Buffer

TLB is a special component of the advanced processors equipped with memory-management unit (MMU), which allows to achieve benefits of paging. It is located on the data link between processor core and physical memory and serves as a cache for virtual to physical addresses mappings (Figure 1). On architectures with physically addressed memory caches TLB precedes the cache on that data link. In contrast, on architectures with virtually addressed memory caches TLB follows the cache. In contrast to the regular CPU caches, TLB is not coherent. Due to this operating systems are forced to synchronize TLBs in multiprocessor systems explicitly. The majority of advanced processors that can be found in desktops, laptops, servers, smartphones and other complex computation devices include one or more TLBs as a part of the memory management facilities providing virtual memory support.

The TLB is almost always implemented as an associative cache. It contains a set of slots, each of which can store a pair consisting from virtual address and associated with it physical address. TLB performs permissions checks and replaces virtual address used by CPU, by respective physical address used by memory controller in each memory access request generated by processor. Each such request can found TLB in two states. In the first state, called hit, TLB contains a mapping for address specified in the request. In the second state, called miss, TLB does not contain appropriate address translation for the address specified in the request. To handle miss translation lookaside buffer per-

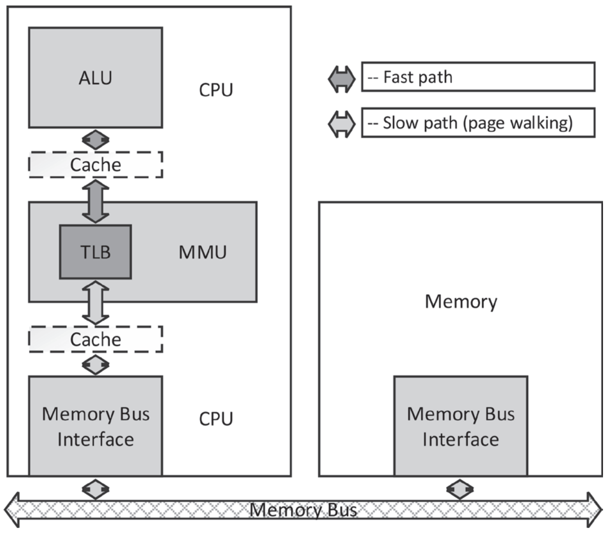

Figure 1. TLB

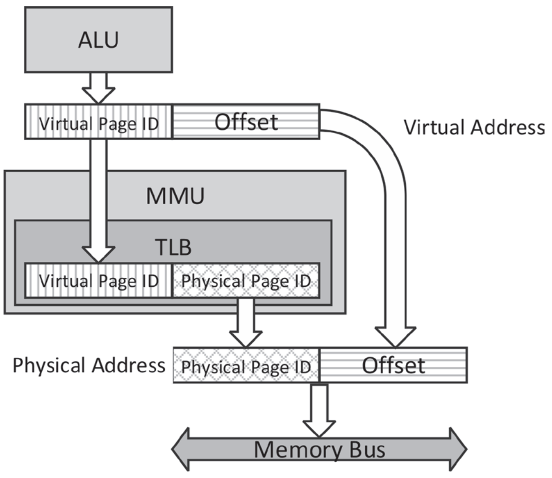

Figure 2. Address Translation

forms costly page walk. It queries page tables stored in main memory performing lookup of the translation requested by operation that has triggered miss. Finally mapping found in page tables is cached in TLB and handling of the original memory operation is restarted.

The virtual and physical address spaces are split into pages with the same fixed size. Every memory address consists of two parts: page identifier and offset. The job of MMU is to translate page identifier from the form of the virtual page id to the form of the physical page id (Figure 2). The page table is stored in main memory and it tracks which physical memory page and to which virtual memory page is mapped. In the case of TLB miss the processor MMU is forced to perform as many additional memory accesses as many levels in the page tables hierarchy exist [1]. Thus, TLB miss leads at least to the doubling of actual memory accesses, and the more levels in the page tables hierarchy, the more performance penalty. For example, on the AMD64 TLB miss is forcing MMU to perform 3 additional memory accesses.

Translation Lookaside Buffer management on the IA-32 platform

During the management of virtual address spaces of processes executing in the system, as well as, management of kernel address space, operating system should synchronize hardware and software views of address space layout. This synchronization should be performed by operating system kernel explicitly to overcome unpredictability and indeterminacy related to the TLB and force MMU to employ up to date translation rules. Note, that modification of page tables in memory does not leads to automatic update of the cached in TLB translation rules.

Translation lookaside buffer provides four kinds of interfaces (Figure 3):

- *phy_address_t Translate(vrt_address_t va, attributes_t operation_attributes)* – performs a physical to virtual address translation.
- *void Load(vrt_address_t va, phy_address_t pa, attributes_t permissions)* – loads translation rule from page table into TLB.
- *void Drop(vrt_address_t va)* – drops one of the cached translation rules from the TLB.
- *void Reset(void)* – resets all translation rules cached in TLB.

Let's consider management of TLB on the example of IA-32 architecture. IA-32 architecture exploits separate TLBs for data and instructions and for 4Kb and 4Mb pages [3].

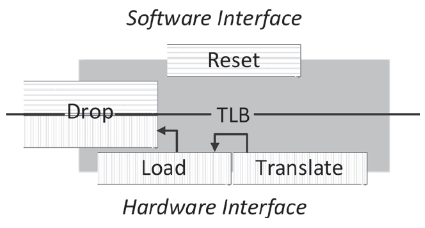

Figure 3. Abstract model of TLB interfaces

TLB designed to be as much automatic as possible. Most frequent operations (*Load*, *Translate* and *Drop*) it performs without involvement of the operating system. Furthermore MMU does not export to the OS interface to the *Load* and *Translate* operations. On the other hand, TLB managemet can not be completely automatic and OS kernel need to have some means of TLB management to modify page tables and switch between virtual address spaces safely. To fulfill that requirement TLB exposes interface to the *Drop* and *Reset* operations to the OS kernel. Note that while Reset operation can be triggered only by kernel, the *Drop* operation can be requested by OS, as well as, by TLB hardware itself in reply to overflow.

IA-32 Instruction Set Architecture exports two TLB management related instructions. First of all, *mov CR3, reg*, which primary purpose is to switch processor from one virtual address space to another. This instruction has a side effect of resetting of the entire content of TLB. In fact, pure *Reset* can be expressed by pair of instructions *mov reg, CR3 | mov CR3, reg*. When *mov CR3, reg* instruction provides a coarse-grain control on the TLB, another instruction – *invlpg reg* provides a means of fine-grain control and implements *Drop* operation which invalidates only one specified translation rule.

TLB management

Typical design of TLB management subsystem of the OS kernel pursues the following objectives:

- Assure predictability and consistency of the virtual address space layout modifications and of the memory access operations behavior.
- Minimize amount of TLB misses.
- Minimize amount of performed TLB management operations.

The main objective of TLB management subsystem of the OS is providing of consistent and predictable memory model for the rest of software executing atop of operating system. From the kernel viewpoint every modification of (write in) page table is a potential source of misalignment of virtual address space in TLB and in page tables. Thus, each such operation should be carefully checked either at design time or at execution time and be accompanied by respective TLB *Drop* if needed. Hence, TLB management subsystem should encapsulate access to page tables and pass reads unconditionally, but provide special processing for writes (Figure 4). There are two optimization tricks that can be applied on the writes.

First of all, it can be taken into the account that virtual address space switch as a side effect leads to entire TLB reset. That mean that there is not any benefit from the explicit invalidation of translation rules related not to the currently active virtual address space. All such changes made in the context of virtual address space *A* in the page tables of virtual address space *B* can be transparently accumulated without explicit synchronization with TLB, because all they will be enforced in batch by processor during the switching to the virtual address space *B*. Taking into the account the fact, that such changes can introduce any observable effects only in the context of virtual address space *B*, such lazy enforcement of translation rules modifications is completely safe from the point of view of consistency and predictability.

Second trick relies on the known fact about TLB implementation on IA-32 platform. Particularly, on the fact that IA-32 TLB can cache only valid address translations. Attempts to access a memory page that is marked as not present leads to page fault without caching of respective page table entry. This feature leads to the two TLB management subsystem design features. On the one hand, any change in the page table entry which transforms a page from present to not present state must be accompanied by the TLB *Drop* of the respective translation rule. But on the other hand, any change in the page table entry which transforms a page from not present to present state is safe, because it is guaranteed that respective mapping is absent in TLB, and consequent *Load* operation will capture the new translation rule.

Simple TLB management model

The described above observations bring us to the Simple TLB Management Model depicted on the Figure 5. Circles denote the state of page table entry and its current value, while arrows denote transitions between states triggered by memory management operations on the page table entry. Dashed lines on this diagram represent transitions between states of page table entry for which *Drop* of respective TLB entry is redundant and can be omitted without compromising of memory image consistency. Solid lines represent transitions that is necessary to be complemented by *Drop* operation to keep memory image consistent. It can be noted that *Drop* is necessary only when page table entry is transformed from present state and when transformation affects at least one TLB tracked bit in page table entry. Take a note that on the example of IA-32 platform, the page table entry contains three bits [9–11] which are ignored by TLB hardware, thus their

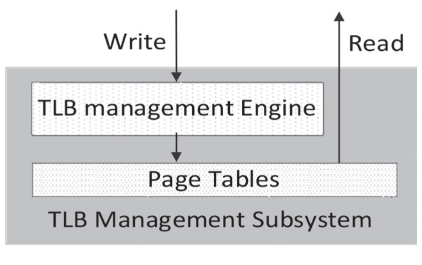

Figure 4. Architecture of TLB Management Subsystem

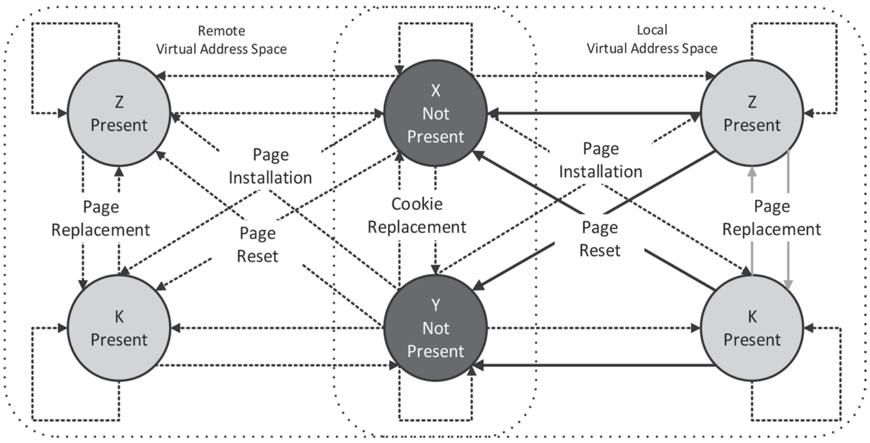

Figure 5: Simple TLB Management Model

change does not affect the translations of virtual to physical addresses.

This model is simple but provides a complete and consistent TLB management policy which can serve as a ground foundation for the design of memory management subsystem of the operating system kernel. In particular, its application to the memory management subsystem design leads to the optimization of kernel code via avoiding of valueless condition checks and invalidation's of TLB entries, and at the same provides a guarantee of memory layout consistency. Finally it allows to abstract out the rest of the kernel from the TLB architecture details by separation TLB management into the distinct layer used by rest of the kernel subsystems (see Figure 4). We argue that due to its simplicity, minimality and the concise nature, application of that model in the domain of true microkernels is especially beneficial. Moreover, Simple TLB Management Model could be used for the purposes formal verification of already existing kernels.

Simple TLB management model was used as a ground model for implementation of M-M/S-CD memory management subsystem for microkernel used as a core component of experimental multikernel OS [7].

Conclusions and future work

We have provided an overview of TLB, including a description of its structure, properties, place, and role in the computer system.

TLB requires special care from the operating system kernel to provide consistent virtual memory layout and deliver a maximum of efficiency. This paper highlights the main challenges related to TLB management, including their description, proposed solutions, and optimization tricks using the example of the IA-32 platform.

We have proposed a simple TLB management model that provides a concise and consistent policy for TLB management and which relies on the availability of means of fine-grained resetting of TLB entries. The introduced model focuses on modifications of the currently active virtual address space. Alterations of external address spaces do not require explicit clean up of TLB buffer, because the virtual address space switch performs a reset of TLB as a side effect.

REFERENCES

1. **Intel** Corporation. IA-32 Intel Architecture Software Developer's Manual. Volume 3: System Programming Guide. 245472–007. 2002.

2. **Seal D.** ARM Architecture Reference Manual. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd ed., 2000. ISBN 0201737191.

3. **MIPS** Technologies. MIPS32 Architecture for Programmers Volume III: The MIPS32 Privileged Resource Architecture, 0.95 ed., 2001.

4. **Bryant R., O'Hallaron D.** Computer systems: a programmer's perspective. Prentice Hall, 2003. ISBN 9780130340740.

5. **Mosberger D., Eranian S.** IA-64 Linux Kernel: Design and Implementation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001. ISBN 0130610143.

6. **Gorman M.** Understanding the Linux Virtual Memory Manager. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131453483.

7. **Klimiankou Y.** M-M/S-CD Memory Management: Conceptual and System Models – In: 2017 Ivannikov ISPRAS Open Conference (ISPRAS), 2017 – pp. 51–57.

Поступила 08.09.2019 *После доработки 08.11.2019* *Принята к печати 01.12.2019*

КЛИМЕНКОВ Е. И.

УПРАВЛЕНИЕ БУФЕРОМ АССОЦИАТИВНОЙ ТРАНСЛЯЦИИ

Белорусский государственный университет информатики и радиоэлектроники

В данной статье основное внимание уделяется управлению Буфером Ассоциативной Трансляции (БАТ) как одному из основных разделов управления памятью в компьютерных системах. БАТ является ассоциативным кешем, включаемым в состав развитых микропроцессоров, для сокращения накладных расходов на отображение адресов виртуального адресного пространства на адреса физического адресного пространства. В предлагаемой работе рассматриваются вопросы, связанные с проектированием подсистемы управления БАТ в ядрах операционных систем на примере платформы IA-32, и предлагается простая модель полной и целостной политики управления БАТ. Предлагаемая модель может быть применена как в качестве основы для проектирования подсистем управления памятью в ядрах операционных систем, так и для верификации таких подсистем в уже существующих ядрах ОС.

Ключевые слова: *виртуальная память, физическая память, управление памятью, управление буфером ассоциативной трансляции.*

Klimiankou Y. I. received the Software Engineer and the MSc degree in the field of «Computers, Computer Systems, and Network Software» from the Belorussian State University of Informatics and radioelectronics in 2010 and 2011 respectively. He is currently working toward a PhD degree in the field of «Computers, Computer Systems, and Network Software» by researching the operating systems design and implementation. His research interests include operating systems, virtualization technologies, virtual execution environments, compilation, and optimization technologies. E-mail: klimenkov@bsuir.by.