M-M/S-CD Memory Management: Conceptual and System Models

Yauhen Klimiankou
Department of Software for Information Technologies
Belarusian State University of Informatics and Radioelectronics
6 P. Brovki Street, Minsk 220013 Belarus
Email: klimenkov@bsuir.by

***Abstract*—M-M/S-CD (Master–Move/Slave–Clone:Destroy) is a conceptual memory management model that follows the minimality principle and it is specially designed for true microkernels.**

M-M/S-CD provides a minimal abstractions for management of both physical and virtual memory of computer system and creates a fundamental and well-isolated memory management layer over the Instruction Set Architecture (ISA) Abstraction Layer. Full set of user space memory management servers can be built on the basis of this memory management layer. Furthermore, the rest of the microkernel subsystems can be designed and implemented using M-M/S-CD as a foundation. Thus M-M/S-CD covers the kernel memory management alongside with user space memory management. In this paper, we discuss the conceptual model of M-M/S-CD and its system model, which highlights implementation details and its linkage with other kernel subsystems. We present the most important features of M-M/S-CD design, such as facilities provided for physical and virtual memory management, isolateness of the physical memory system, in-place memory accounting and more. We also discuss the various design decisions that influenced the implementation of these features on the IA-32 platform. We conclude by describing the current implementation status and plans in regard of M-M/S-CD.

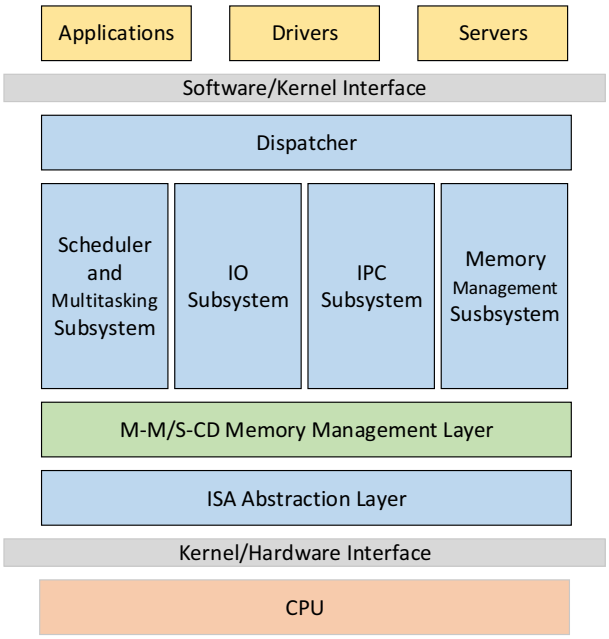

Figure 1. *Functional View of M-M/S-CD*

1. Introduction

M-M/S-CD was designed and developed as a part of the academic research project at the intersection of multikernel [1] and microkernel [2] operating systems design and implementation. This research is based on the operating system written from the scratch, kernel of which is designed atop of M-M/S-CD memory management model. M-M/S-CD is implemented as a C++ component with about 1500 code lines, which compiles in about 1KB of binary code.

The current version of M-M/S-CD is used as a platform for our research microkernel in the prototype of multikernel/microkernel operating system for the IA-32 platform [3].

M-M/S-CD offers a combination of features to achieve the overall goal: provide the flexible and powerful foundation for building the OS, while being simple and minimalist to be suitable for microkernels. Figure 1 illustrates a functional view of M-M/S-CD in the kernel.

2. Motivation

Memory and processor time are the main computational resources provided by the computer systems. Due to this, scheduling and memory management always aroused interest of the operating systems designers and developers [4]. Different types of computer systems can impose different restrictions and requirements to the scheduling and memory management. For example, try to consider and compare a high-performance server in the cluster and the industrial embedded computer system of hard real-time. In order to provide standardized, flexible and powerful memory management architecture and to minimize the efforts of engineers in development of new kernels, and to maximize flexibility of the memory management policies (to allow implementation of such unusual policies as self-paging [5]) as well as scalability of the memory management subsystem, a new memory management model is required. Such model

0-7695-6356-2/17/$31.00 ©2017 IEEE
DOI 10.1109/ISPRAS.2017.00016

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

should also support different operating system architectures, including operating systems based on monolithic kernels and microkernels, multikernel operating systems, as well as operating systems of hard and soft real-time.

2.1. Existing Approaches

Recursive address space construction (RASC) is the original approach to the memory management proposed by Liedtke in his L4 microkernel [6]. This model is conceptually simple and elegant. It is based on an idea that process has management rights for physical memory which has mapped into the virtual address space (VAS) of the process. Thus, RASC model creates a tree-like hierarchy of physical memory derivations using three basic operations provided: Map, Grant and Flush. Special VAS σ_0 is used as a container for all physical memory available in the system and, at the same time, as the root of the RASC tree. Substantial book-keeping in the form of a mapping database is required in order to support the revocation operation with the page granularity. The most significant shortcomings of RASC are the following [7]:

- Book-keeping in the form of a mapping database consumes significant amount of memory. Actually, it was found that in many cases this database becomes a major kernel memory consumer.
- RACS in general case requires implementation of a channel for physical memory swapping from user space into kernel and back.
- Absence of appropriate isolation of memory resources makes RASC vulnerable for Deny-of-Service attacks in the form of continuous cyclic mappings of the same physical memory between two or more processes.
- Memory operations in RASC are indeterministic (especially revocation). As a result, RASC does not fit well into real-time systems.

Capability-Based Memory Management (CBMM) is based on the ideas of grant-take model [8] that was proposed as an alternative to the RASC [9]. CBMM is based on ideas of EROS [10] and is focused on security, protection and strict isolation of the memory and performance between processes. This model proposes the idea to make all objects of kernel explicit. Although CBMM overcomes a lot of disadvantages of RASC it is still based on hierarchical book-keeping in form of capability derivation tree [11]. Capability derivation tree controls creation and destruction of all kernel objects in the system and provides means for recursive revocation of memory. CBMM moves big part of memory management from the kernel into user-space and requires user-level book-keeping.

Both RASC and CBMM are general purpose memory management models used in L4-derivatives nowadays. And both models are hierarchical which puts substantial book-keeping into the kernel responsibilities to provide the ability of revocation of memory and thus assure integrity of memory system on the cost of additional complexity.

2.2. Assumptions

The M-M/S-CD design is based on the following assumptions, known to be valid within most ISAs of advanced processors:

- Target processor contains MMU.
- Memory management is based on paging, not on segmentation.
- Mappings are performed through potentially hierarchical page tables.
- Memory management is performed in terms of continuous fixed-size pages.
- Page faults are supported by ISA.
- VASes are supported by ISA.
- Efficient support of wide variety of user-level memory management services and policies is important in the modern OSes.
- The main objectives of memory management subsystem of the kernel are assurance of integrity of memory system and isolation of memory between kernel space/user space and between processes.
- The auxiliary goals of M-M/S-CD design are simplicity, minimalism and determinism.
- All dynamic kernel objects can be expressed as a data structures with sizes a multiple of memory page size.

Because of the above reasons, M-M/S-CD was designed as a flat model in contrast to hierarchical models of RASC and CBMM. It requires a minimal amount of book-keeping in the kernel (heapless kernel design) and thus minimizes the amount of used data structures (on IA-32 it reuses the page tables without introduction of additional data structures at all). This in turn gives a lot of other advantages, such as flexibility, simplicity and minimalist implementation.

3. M-M/S-CD architecture : conceptual model

M-M/S-CD shares the ideas proposed in Chorus [12] and considers memory management as organized in a layered fashion (Figure 2). M-M/S-CD completely encapsulates TLB management and provides building blocks for VAS layout construction and for creation of tools for memory management and facilities, used by other kernel subsystems. Subsystems of user space memory management and inter-kernel communication are the main users of M-M/S-CD. User space memory management subsystem of the kernel exports access to the conceptual M-M/S-CD model for the user mode memory management servers. Thus, all user space memory management functionality and policies are completely moved from the kernel into user mode. Inter-kernel communication subsystem is an extension of microkernel that is responsible for dynamic intra-node memory exchange in the multikernel operating system.

M-M/S-CD considers physical address space as an array of pages of the same size (4KB on IA-32 platform).

M-M/S-CD is based on two basic abstractions: physical memory page and its container (slot in VAS). Container

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

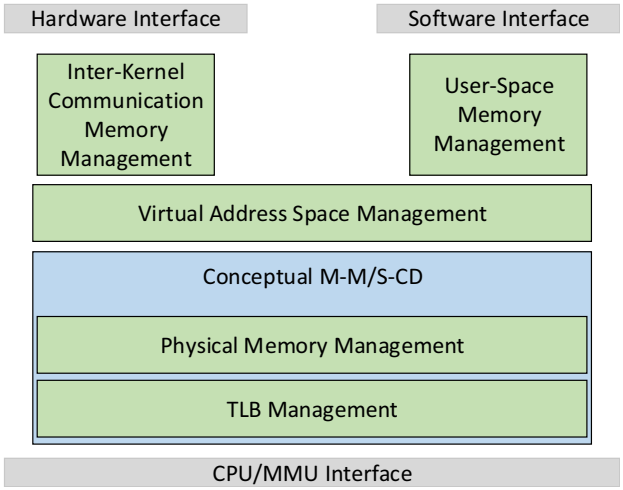

Figure 2. *Layered View of M-M/S-CD model*

can be either empty or contains a reference to the physical memory page. There are two kinds of references that can be stored in container: master and slave. Each physical memory page, available on computer system and managed by microkernel, is referenced by exactly one master reference which provide management and access rights to that page. At the same time each physical memory page is able to have variable number of slave references (starting from 0 and up) which gives only access rights. If the page does not have slave references, it is considered as being in unladen state. Each physical memory page has a slave reference counter assigned to it At the same time, each container includes a type indicator for the reference stored in it.

There are three operations supported by M-M/S-CD on containers: move, clone and destroy (Fig. 3). Movement is allowed only from container holding master reference to unladen page into empty container. Move operation does not affect reference counter of moving page. Page cloning is allowed only from container holding master reference into empty container. Clone operation increments reference counter of the cloned page. And finally, page destruction is opposite – it is allowed only for container holding slave reference and it switches container into empty state. Respective reference counter is decremented. To summarize, master references can only be moved, while slave references can only be created and destroyed.

M-M/S-CD states that non-empty container can not be destroyed. This fact, in conjunction with a fact that master references is only movable, provides a guarantee that no physical memory pages can arrive into system at runtime and no memory pages can leak from the system. Hence, M-M/S-CD creates a closed memory system without any special memory state tracking data structures like σ_0 address space in RASC or capability-derivation tree in CBMM. In both cases, data structures based on interval tree are used, where root of the tree is used to provide a guarantee of the integrity of physical memory layout.

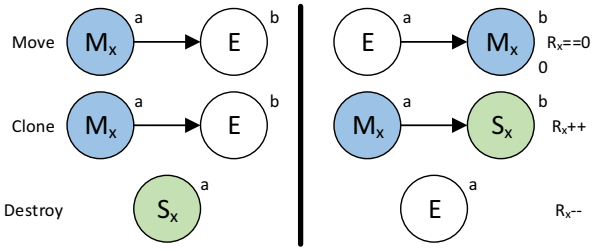

Figure 3. *Conceptual Model of M-M/S-CD*

4. M-M/S-CD architecture : system model

A complete full-featured memory management subsystem of microkernel can be designed based on the pure conceptual model of M-M/S-CD. However, this approach will lead to inconvenient interfaces for the rest of the kernel subsystems and, what is more important, to the suboptimal performance of all memory related operations. Due to this, refined system model of M-M/S-CD was designed which accounts different aspects of typical intra-kernel memory usage and exploits them to optimize performance. This model is described below.

Containers of M-M/S-CD were designed to fit into entries of page table used by processor MMU to perform virtual-to-physical address translation. Thus, page tables can be viewed as an array of page reference containers. Conceptually, there are two families of container types: physical and virtual containers. Physical containers are those located in the page tables used by MMU for address translation. In contrast, virtual containers are used for memory tracking purposes only, hence TLB management enabled for them is valueless.

Note, that in contrast to the RASC and CBMM which are based on usage of special interval-tree-like data structures for memory accounting, M-M/S-CD shares page tables with MMU and actually performs in-place memory accounting. One of three unused bits in the IA-32 page table entry is used for encoding of reference type. Page reference counters are organized into sparse 4Mb table, where index of the page in the physical address space also works as an index in page reference counters table.

System model of M-M/S-CD currently supports the next types of refined virtual containers: Proto Control Block, VAS Control Block, Inter-Kernel Communication Control Block, Inter-Kernel Communication Reserve Page, Physical Dump and Virtual Dump. Supported physical containers include: User Memory Page, Fixed Memory Page, Thread Control Block, Zone Control Block and Window Control Block. 11 types of containers in total, where 5 of them represent dynamically allocated kernel objects, another 4 are thread-local and are used to facilitate access to the kernel data for the core kernel subsystems, another one is used to protect kernel code and static data, and the last one – for user mode applications memory management. Figure 4 demonstrates system model of M-M/S-CD in the form of the transition diagram.

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

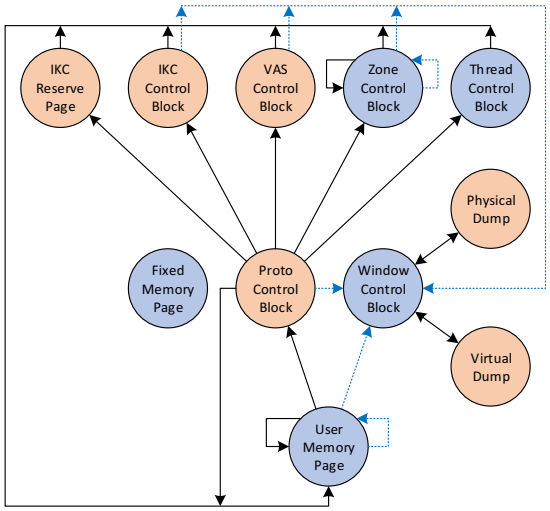

Figure 4. *System Model of M-M/S-CD*

Each type of physical container enforces a particular set of the page table entry flags on IA-32 for the referenced physical memory page. These flags are summarized in Table 1. Other properties of M-M/S-CD containers are summarized in table 2.

TABLE 1. PTE FLAGS ON IA-32 FOR M-M/S-CD CONTAINERS

Property	M	G	PCD	PWT	U/S	R/W
Fixed Memory Page	V	V	?	?	X	?
Thread Control Block	V	V	X	X	X	V
User Memory Page	?	X	?	?	?	?
Window Control Block	X	X	X	X	X	V
Zone Control Block	?	X	X	X	V	X

TABLE 2. PROPERTIES OF M-M/S-CD CONTAINERS

Property	FMP	ICB	IRP	PCB	PD	TCB	UMP	VCB	VD	WCB	ZCB
Physical	V	X	X	X	X	V	V	X	X	V	V
Local	X	X	X	V	V	X	X	X	V	V	X
Movable	X	V	V	V	V	V	V	V	V	V	V
Clonable	X	X	X	X	X	X	V	V	X	X	V
Mappable	X	V	X	V	X	X	V	V	X	X	V
Protected	V	V	V	V	V	V	X	X	V	V	X

User Memory Page (UMP) container exactly repeats a container from conceptual model of M-M/S-CD and is used for representation of user mode VAS and for intra-process and inter-process memory management.

On the opposite side, fixed memory page container holds a master reference and all operations are forbidden for it. Such containers are used to create constant part of kernel VAS that contains kernel code and static data (GDT and IDT tables for example).

Note that the classical microkernel operating system design does not require support of all 11 types of containers. The only dynamic kernel objects supported by classical microkernel-based operating system are threads and virtual address spaces. As a result, IKC reserve page and IKC control block are required in the case of multikernel operating system. Similarly, Physical dumps are required only for the cases when support of inter-address space are required. For example for implementation of asynchronous message passing.

5. Proto Control Block (PCB)

To assure isolation between user space and kernel space, only 'move' semantics is allowed for the inter-space memory exchange. Each freshly-arrived into kernel page is used to build a new kernel object. To eliminate the risk of potential race conditions on partially initialized kernel object the concept of PCB was introduced.

PCB is a thread-local virtual container with a single kernel invocation lifetime. It is created only by reference movement from the UMP. At the moment of the return to the caller from the kernel invocation, reference in PCB should be either moved into one of other containers in the kernel domain or should be returned back into the original UMP container. Such back path is required to handle system call failures. To assure availability of back path, original container is locked at the moment of reference movement into PCB, and is unlocked only when PCB is destroyed during exit from the kernel invocation. Figure 5 represents a kernel object lifecycle.

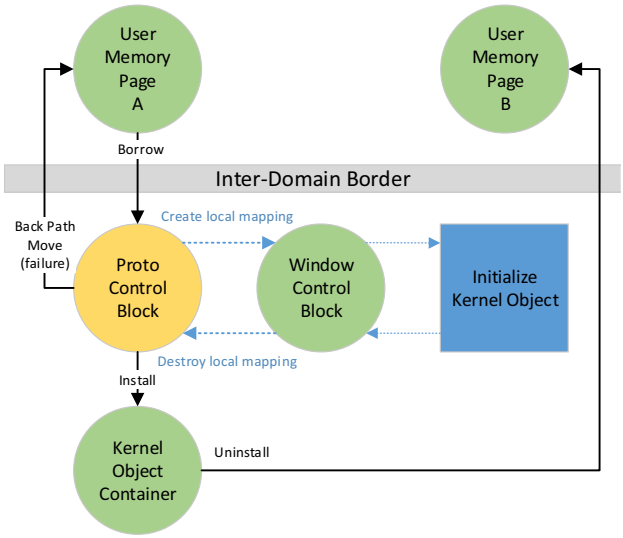

Figure 5. *Kernel Object Lifecycle in the M-M/S-CD model*

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

6. Thread Control Block

Threads are the one of the major dynamic kernel objects. Thread Control Blocks are used to build threads table, that is always mapped in the kernel address space which in its turn is shared between all VASes of the system. Memory pages referenced by thread control blocks are always unladen. Further more thread control blocks support only move-semantics and implement ownership protection. Ownership protection makes thread control block accountable resource of the system that in its turn allows to make thread creation and destruction operations fast, predictable and deterministic.

7. VAS Control Block

VAS Control Block, in conjunction with Thread Control Block, is a main building block of the process abstraction. It is virtual container with additional facility, that allows activation of the referenced page as root of the active VAS. For this container movements is allowed only from PCB and into UMP. At the same time, movement to UMP is conditional and actually performed only for unladen pages. Cloning semantics is modified and allows to clone reference only into another VAS Control Block and in contrast to the original Clone semantics creates not slave, but another master reference. As a result, all threads of the process can share VAS with equal rights to create and destroy another threads. In the case of the original semantics of the conceptual M-M/S-CD model there should be one "main" thread of the process. It is evident, that such behavior will seriously limit the design space of the user-level threading.

8. Zone Control Block

Zone control block represents VAS region with a fixed size. On IA-32, Zone Control Block corresponds to page directory. Zone Control Block container is very similar to the UMP container with two important differences. The first one – movement is allowed only from PCB and into UMP, but not between different Zone Control Blocks. The second one is a modified method of page reference accounting. Each time when UMP container of a region described by Zone Control Block is filled by reference, the reference counter of Zone Control Block is incremented. And vise versa, when reference from UMP container is released, the reference counter of Zone Control Block is decremented. Both this differences assure that zone always arrives into VAS with empty UMP containers and can be removed from VAS only when all its UMP containers are empty. Thus, one of the rules of the conceptual M-M/S-CD model is enforced.

9. Support of Inter-Kernel Communication

Multikernel operating system is designed as a logical network of interconnected and isolated from each other "virtual computers", where each computer has its own CPUs (processor core of cores) and its own memory. Hence, memory layout of each "virtual computer" should stay closed, like in the case of ordinary microkernel OS. But to make the dynamic int er-kernel communication and the inter-kernel memory exchange possible this closeness should be violated in controllable way. Due to this, IKC Control Block was introduced to provide a way of direct setup of the reference value and its unconditional removal. Reference counter of the directly set page can be located in absent memory page. Similarly, when reference is directly removed and there is no other containers referencing that page, it should be released. To solve that issue, IKC Reserve Page was introduced. It serves as a container for memory reserved for IKC operation to allow transparent arrival and departure of physical memory.

10. Inter-VAS Memory Access

Window Control Block implements a form of the Thread Local Storage and is used for two purposes. First one is an implementation of temporal, local and isolated mappings of the page referenced in the virtual container into the kernel part of thread address space. Second one is an establishing of temporal mapping for the memory pages belonging to the foreign VASes. That can be required for inter-VAS data copying, for example, for asynchronous message passing or large message exchange implementation. We distinguish two types of dumps (physical and virtual) because they implements different policies in regard of TLB.

11. Performance

Measurements were done on IBM x335 server equipped by two Intel Xeon CPUs (Netburst/Prestonia 2.8/0.4GHz L1=12K μops + 8KB L2=512KB). One CPU in use, Hyper-Threading was disabled. The performance figures that we have are obtained using timestamp-based microbenchmark in the our experimental multikernel operating system and in the seL4-based operating system.

We measured the execution time taken by each of the M-M/S-CD memory management operation in our operating system. For the purposes of comparison, we measured analogs of the M-M/S-CD operations in the seL4. As a counterpart of Clone operation we have taken an optimized pair of operations seL4_CNode_Copy() + seL4_x86_Page_Map() (as specified in the seL4 reference manual [13]) and an original interface sel4utils_dup_and_map() found in libsel4utils. Both operations shares memory between two virtual address spaces. Similarly, as a counterpart of Destroy operation we measured the execution time taken by an optimized pair of operations seL4_x86_Page_Unmap() + seL4_CNode_Delete() [13] and an original interface sel4utils_unmap_dup() from libsel4utils. Both procedures revokes previously established sharing of memory. Finally, as a counterpart of Move operation in seL4 we considered as a pair of Clone and Destroy operation analogs.

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

Take a note, that memory management operations in seL4 has its own features. CBMM systems like used in seL4 rely to the generalized abstraction of resource – capability. Due to this, in contrast to M-M/S-CD, in CBMM the ownership on the access rights to the resource does not mean that this resource is in use. As a result, process is able to own memory, which is not mapped into its virtual address space. In contrast, in M-M/S-CD ownership means also an usage. This feature of CBMM highlights why two system calls actually needed to share memory in seL4. The first call (seL4_CNode_Copy()) passes access rights to the memory resource (via making a copy of capability) to the specified process. And second call (seL4_x86_Page_Map()) finally maps the memory described by capability into the virtual address space. (Note that capability can be mapped only once. To make a second mapping (achieve memory sharing) an unused copy of original capability is required.) The similar logic is applied for revocation of memory sharing.

Each operation was performed and measured 10 000 000 times. Cache warm up was performed before each series of experiments. Results were normalized to eliminate overhead introduced by measurements itself. Table 3 shows average measured execution time for each M-M/S-CD operation and its counterpart in seL4. Table 4 demonstrates average execution time for each mentioned system call in seL4.

TABLE 3. EXECUTION TIME FOR M-M/S-CD MEMORY MANAGEMENT OPERATIONS

Memory management operation	M-M/S-CD inter-space [ticks]	M-M/S-CD intra-space [ticks]	seL4 (copy&map) [ticks]	seL4 (dup&map) [ticks]
Clone	562	562	17303	18421
Destroy	512	1100	31171	31909
Move	573	573/1166	48382	50309

TABLE 4. EXECUTION TIME FOR SEL4 MEMORY MANAGEMENT RELATED SYSTEM CALLS

System call	execution time [ticks]
seL4_CNode_Copy	5560
seL4_CNode_Delete	17605
seL4_x86_Page_Map	11508
seL4_x86_Page_Unmap	13225

The overall result shows that M-M/S-CD is dramatically faster than CBMM implemented in seL4 microkernel. Most of the overhead in the case of CBMM is introduced by kernel itself. Even extremely optimized analogs of Clone and Destroy operations in CBMM are extremely slower than their counterparts in M-M/S-CD. Note that in both cases operations were performed on the same memory pages with size of 4Kb. Also note, that our implementation of M-M/S-CD can still be underoptimized (because kernel is still in the active development), while seL4 can be considered as a high-quality extensively optimized microkernel.

12. Implementation

12.1. Source Code

Current implementation of M-M/S-CD (with inclusion of ISA abstraction layer) is written in C++03 and consists of 808 lines of source code (+704 lines of ISA abstraction layer) and compiles into 1053 bytes of binary code. Most of the code implementing M-M/S-CD is located in header file in the form of the inline functions and is used mostly to enforce appropriate semantics related to the memory management abstractions. In fact, united together ISA abstraction layer, M-M/S-CD memory management layer and dispatcher layer form a usable framework for microkernel development by efficient separation of processor-specific code from the rest of kernel logic.

12.2. Standardization

At this moment M-M/S-CD is in the ready-to-use state. Nevertheless, while core principles behind M-M/S-CD have proven to be right, there is still a concern about interface provided by M-M/S-CD to the other kernel subsystems. We feel that that interface is not matured enough yet, and expect that it can be changed, extended or reduced in the future. Once it is evident that the interface is stable, it will be formalized and standardized to facilitate and simplify future developments and researches, which will rely on the M-M/S-CD memory management layer.

12.3. Roadmap

Most of the works related to the M-M/S-CD are finished at this moment. We have a microkernel successfully built atop of M-M/S-CD layer. But system model of M-M/S-CD can yet be reduced by elimination of IKC Reserve Blocks during the planned redesign of inter-kernel communication. Also we have a plan to build a set of different kernels atop of M-M/S-CD to achieve virtually heterogeneous multikernel operating system.

13. Conclusion

M-M/S-CD has proven to be a good and useful foundation for design and implementation of microkernels. Furthermore, M-M/S-CD provides a flexible, powerful, extremely efficient and in the same time minimalist model of memory management. Major memory management operations in M-M/S-CD are 30-60 times faster than their analogs in CBMM. It is our hope that the ideas behind M-M/S-CD will be reused by others to build various kinds of microkernels targeted to different types of operating systems.

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.

References

[1] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: A new os architecture for scalable multicore systems," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 29–44.

[2] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A new kernel foundation for unix development," 1986, pp. 93–112.

[3] Intel Corporation, *Intel Architecture Software Developers Manual Volume 3: System Programming*. Santa Clara, CA, USA: Intel Corporation, 1999.

[4] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

[5] S. M. Hand, "Self-paging in the nemesis operating system," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 73–86.

[6] J. Liedtke, "On micro-kernel construction," in *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, SOSP 1995, Copper Mountain Resort, Colorado, USA, December 3-6, 1995*, 1995, pp. 237–250.

[7] K. Elphinstone and G. Heiser, "From l3 to sel4 – what have we learnt in 20 years of l4 microkernels?" in *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, Farmington, PA, USA, November 2013, pp. 133–150.

[8] R. J. Lipton and L. Snyder, "A linear time algorithm for deciding subject security," *J. ACM*, vol. 24, no. 3, pp. 455–464, Jul. 1977.

[9] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "sel4: Formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 207–220.

[10] J. S. Shapiro, J. M. Smith, and D. J. Farber, "Eros: A fast capability system," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 170–185, Dec. 1999.

[11] D. Elkaduwe, P. Derrin, and K. Elphinstone, "A memory allocation model for an embedded microkernel," in *In 1st MIKES*, 2007, pp. 28–34.

[12] E. Abrossimov, M. Rozier, and M. Shapiro, "Generic virtual memory management for operating system kernels," in *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, ser. SOSP '89. New York, NY, USA: ACM, 1989, pp. 123–136.

[13] *seL4 Reference Manual. Version 7.0.0*, Data61 Trustworthy Systems, Trustworthy Systems Team, Data61, 2017.

Authorized licensed use limited to: Zhejiang Tmall Technology Co.Ltd.. Downloaded on July 15,2020 at 08:41:41 UTC from IEEE Xplore. Restrictions apply.