КРАТКИЕ СООБЩЕНИЯ

УДК 004.451.3

MODEL OF M-M/S-CD MEMORY MANAGEMENT

Y.I. KLIMIANKOU

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus

Submitted 15 May 2017

Abstract. This report introduces the M-M/S-CD memory management model which was designed for true microkernels as an alternative to capability-based memory management. The proposed model is more simple in design and implementation and provides more power and flexibility to the user mode memory management servers. In the same time it maintains consistency of memory layout and enforces strict and clear memory isolation/sharing rules, creating stable environment for applications.

Keywords: memory management, physical memory, virtual memory.

Doklady BGUIR. 2018, Vol. 111, No. 1, pp. 91-94
Model of M-M/S-CD memory management
Y.I. Klimiankou

Introduction

Memory management is one of the core functions of the operating system kernel [1]. This report serves as a brief introduction into M-M/S-CD memory management model, which was designed specially for operating systems based on the second generation of microkernels. In contrast to the recursive address space construction [2] and capability-based memory management models [3], M-M/S-CD is conceptually simpler both in design and in implementation and covers such domains as user space, kernel space, physical and virtual memory management.

Philosophy

The M-M/S-CD memory management model is designed as a fundamental basis for building true microkernels and operating systems atop of them. Due to this it supports such aspects of memory management as kernel space and user space, physical and virtual memory management. The M-M/S-CD is designed with two key properties in the mind: isolateness, explicitness and relies to the uncommon reasoning about memory management in terms of sharing/isolation instead of allocation/deallocation.

The M-M/S-CD memory management model considers physical memory of the computer as an isolated system. It assumes that quantitative and qualitative properties of physical memory available in the system is constant during entire system lifetime. Physical memory pages can not be added, removed or moved to, from and in the pool of system memory in a run time. Each Physical Memory Page (PMP) in M-M/S-CD is always mapped into at least one Virtual Address Space (VAS). It is the responsibility of bootloader to investigate physical memory layout and to create system memory pool at the operating system initialization time.

The M-M/S-CD exploits the idea of explicitness of all memory management operations. In contrast to the commonplace operating systems, where kernel plays both roles: manager and consumer of memory [4][5], M-M/S-CD considers kernel as a virtual process with the same rights in regards to the memory as any other process in the system. Thus, in each system call, which

servicing requires kernel to use additional memory, application passes the explicitly specified PMPs to the kernel. Similarly, each system call, which servicing releases some memory in the kernel space, returns such memory into location explicitly specified by caller. Kernel itself only performs requested operations assuring consistency of memory layout and enforcing access control. User-space memory management servers, in their turn, are aware about every change in the state of the owned physical memory and virtual address space. Explicitness of memory management operations allows to move all memory management bookkeeping from the kernel space into the user space. Furthermore it leads to the creation of more deterministic, predictable and efficient applications and systems.

In contrast to the traditional reasoning about memory management in terms of allocation/deallocation, M-M/S-CD exploits reasoning in terms of sharing/isolation. Every piece of physical memory is permanently mapped to at least one of the processes, which can either ignore it or utilize it as a storage for data. User space memory management is performed using three basic operations:

– isolation of the memory between VASes;
– sharing of the memory between VASes;
– passing of the memory management rights between VASes.

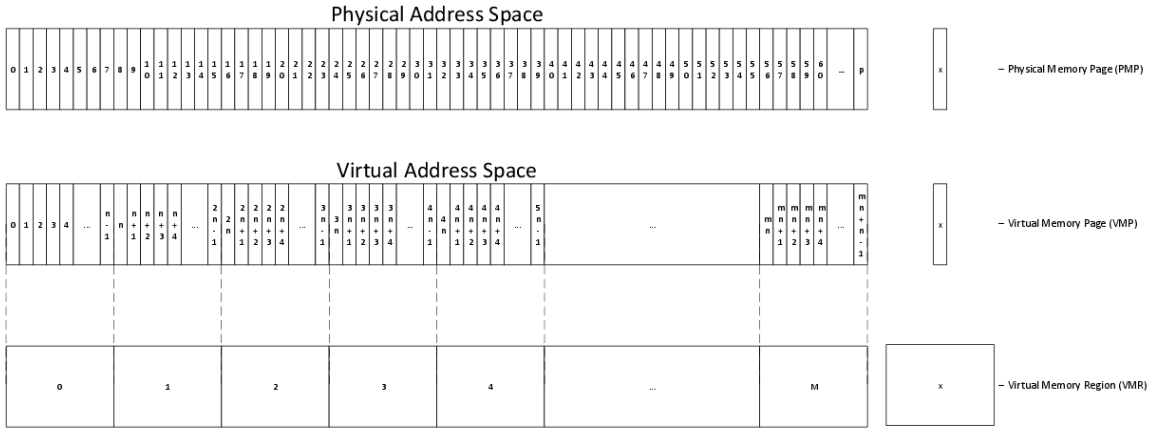
Using these fundamental operations user-space memory management servers are capable to implement memory management services with traditional allocation/deallocation semantics. Indeed, memory allocation can be easily mapped into memory sharing and memory deallocation – into the memory isolation, while passing the memory management rights allows to divide physical memory between multiple concurrently running memory management servers and assure isolation of memory managed by each of them.

M-M/S-CD model covers both domains of memory management: physical and virtual memory management and encapsulates Translation Lookaside Buffer management. Paging and other complex memory management services are implemented out of scope of the M-M/S-CD and on its basis. Furthermore, M-M/S-CD can be subdivided into two sublayers: basic intra-kernel memory management layer on the bottom and user-space memory manager on the top. Bottom layer creates a basis for memory management atop of which all other kernel subsystems are built. Top layer in its turn creates a Memory Management API exported into user space for use in memory management servers.

Abstractions

M-M/S-CD memory management model is designed around four basic abstractions (Figure):
– physical Memory Page;
– virtual Memory Page (VMP);
– virtual Memory Region (VMR);
– \item Virtual Address Space.

PMP serves as a representation of physical memory, while virtual memory page, virtual memory region and virtual address space serve as representation of virtual memory.



M-M/S-CD abstractions

Physical Memory Page. PMP represents continuous region of physical memory with fixed size and properly aligned. Entire physical address space is considered as an array of pages. Every PMP has two attributes: address and reference counter. Note that every PMP represented in the system has at least one reference.

Virtual Address Space. VAS plays a role of proto process as a container for threads and at the same time works as a container for PMPs. At every moment of the system lifetime there is at least one VAS. Note that in the case of only one VAS present in the system, such VAS will contain all physical memory available in the system.

Virtual Memory Region. VAS consists of a number of a fixed size regions. VMR serves as a unit of VAS management and has an assigned owner thread/process. Owner is not necessary to be executing in the same VAS in which VMR is located. Due to this regions provide a mean of external VAS management and, thus, create a basis for pagers and memory managers work.

Virtual Memory Page. Each VMR in its turn consists of fixed number of virtual memory pages. VMP works as a container for PMPs. Each virtual memory page either contains a reference to PMP or can be empty. Note that multiple VMR can reference the same PMP, but for each PMP there is at least one VMP referencing it.

M-M/S-CD abstractions tightly reflect the abstractions of page tables and page table trees which are in use by processor MMUs (for example in IA-32 [6]). This feature of M-M/S-CD memory management model makes it able to get simple implementation, which in many cases does not require usage of additional data structures and extensive tracking of the memory state. If fact, M-M/S-CD is able to reuse page tables used by MMU for memory tracking and management. Simplicity of implementation in its turn leads to better performance, predictability and determinism in memory management operations.

Operations

The key concept of M-M/S-CD memory management model is distinguishing between two types of references to PMPs: master and slave. The main feature of master references is that they can be neither created nor removed. Due to this in every moment of system lifetime for each PMP present in the system there is exactly one master reference. Master reference provides the VAS that owns it and threads running in context of that VAS both access and management rights for the referenced PMP. In addition to master reference, every PMP present in the system can have any number of slave references. In contrast to master references, number of the slave references pointing to the same PMP can change during system lifetime. Slave reference provides to the owner access rights to respective PMP, but does not provide the rights to manage it. Presence of master references provides isolateness of the memory management system.

M-M/S-CD supports three basic memory management operations: Clone, Destroy and Move.

Move(from, to). Where *from* is a VMP which contains a master reference to PMP, which does not have slave references, and *to* is an empty VMP. In case of success, after operation completion *from* will be empty and *to* will contain the master reference originally stored in *from* VMP.

Clone (from, to). Where *from* is a VMP which contains a master reference to PMP and *to* is an empty VMP. In case of success, after operation completion *from* will be unmodified and *to* will contain the slave reference to the PMP referenced by *from* VMP.

Destroy (from). Where *from* is a VMP which contains a slave reference to PMP. In case of success, after operation completion *from* will be empty.

Note that operations *Clone* and *Destroy* modify the reference counter of the respective PMP, while *Move* checks reference counter to validate eligibility of the master reference movement.

The M-M/S-CD model distinguishes two types of the subjects in the memory management: local threads and pager threads. Local thread is any thread executing in the context of current VAS. Pager is a thread, potentially executing in the context of the external virtual address space, but having some rights to manage a region of the current VAS.

M-M/S-CD memory management model employs simple access control policy. Each virtual memory region has an assigned pager. Pager can be assigned and rejected by any thread running in the context of VAS to which region belongs. Pager of the region has rights to create and destroy slave pages in that region, as well as move master pages to that region and from it. At the same time,

threads are allowed to move master pages from any region of the local VAS disregarding the owner of the region.

Any page fault occurred in the virtual memory region is forwarded for handling to the assigned pager, which has all rights necessary to handle fault transparently to the application. At the same time process has full control on its own VAS as a entity which makes decision about which pager and to which VMR should be assigned. Note that multiple memory managers can service the same VAS (process) concurrently, each managing its own VMR.

Conclusions

It's briefly introduced the M-M/S-CD memory management model which provides elegant and complete architecture, that was developed with focus on application in operating systems based on second generation of microkernels. The M-M/S-CD covers both physical and virtual memory management and designed around four basic abstractions and three conceptual operations (clone, destroy and move). The M-M/S-CD provides a good alternative for another memory management models like capability-based memory management and recursive address space construction and leads to more minimalist, efficient, predictable and deterministic solutions, which makes it especially useful for the real-time systems design and implementation.

References

1. Jochen Liedtke. Toward real microkernels. Commun. ACM. 1996. № 39 (9). P. 70–77.
2. Jochen Liedtke. Improving ipc by kernel design. SIGOPS Oper. Syst. Rev. 1993. № 27(5). P. 175–188.
3. Dhammika Elkaduwe, Philip Derrin, Kevin Elphinstone. A memory allocation model for an embedded microkernel // International Workshop on Microkernels for Embedded Systems. Sydney, Australia, January 2007. P. 28–34.
4. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. Linux Device Drivers. O'Reilly Media, Inc., 2005.
5. Mark Russinovich, David A. Solomon. Windows Internals: Including Windows Server 2008 and Windows Vista. Fifth Edition. Microsoft Press, 5th edition, 2009.
6. Intel Corporation. IA-32 Intel Architecture Software Developer's Manual. Vol. 3: System Programming Guide. Number 245472-007. 2002.

Information about the author

Klimiankou Y.I., PG student of information software technologies department of Belarusian state university of informatics and radioelectronics.

Address for correspondence

220013, Republic of Belarus, Minsk, P. Brovka st., 6,
Belarusian state university of informatics and radioelectronics
tel. +375-33-604-51-47;
e-mail: klimenkov@bsuir.by
Klimiankou Yauhen Ivanovich