

Building Flexible OS with Exokernel and Cache Kernel

Rakhim Davletkaliyev

Introduction

Existing operating systems provide an interface for applications and services to run on and have easy and secured access to hardware resources. OS runs as a layer between hardware and software, so every access to those resources have several restrictions and rules; they can be called a policy. Two main types of today's operating systems are monolithic and microkernel, as it seen from the names – the difference is in the structure of the kernel. Both of these approaches use certain policies for application level, usually seeing and making no difference between various programs asking for resources. Therefore, one or few policy schemes are used regardless program specifications, making the whole system less efficient. In many cases applications require far greater control over memory, I/O and processing resources than an OS can provide.

There exist some new approaches for making resource management more efficient on both hardware and operating system level, because large complicated objects of an operating system (kernel functions, address spaces, processes, memory mappings) are usually act as resources themselves. In this paper I'm trying to overview different ideas on building a new type of kernel which deals with different applications and resources in a new manner, giving applications more control over hardware and giving them ability to create their own policies, meanwhile controlling and securing OS resources with a mechanism called kernel cache.

Related work

MIT Laboratory for Computer Science released a paper The Operating System Kernel as a Secure Programmable Machine [1], which talks about importance of exokernel model (Latin "ex" – out of, without), where applications customize OS kernel by extending the exokernel interface. Exokernel provides minimalist architecture for efficient work of several processes. To test and evaluate exokernels and their customization techniques, *Aegis* is being developed [1].

Considering this model, kernel needs more flexible security mechanisms, and at the same time – speed of executing and fault tolerance should be taken into account. Stanford University Computer

Science Department released a paper called A Caching Model of Operating System Kernel Functionality [2]. The main idea is to create a kernel cache, which acts as regular application cache, but instead of application data, operating system resources being cached. As a result, a new type of microkernel is developed, smaller than regular microkernel, and tested on database, real-time and large-scale simulations.

These two ideas are main fundament for this paper; they show importance of efficient memory and OS resource management considering different applications with different need of resources.

Importance in the Context of Operating Systems

History of development of operating systems show us a lot of different approaches and general ideas, but talking about most used regular operating systems of today, in last few years we've seen two main models of an OS kernel – monolithic and microkernel.

Talking about policies for applications to use hardware resources, monolithic kernel usually is least effective. One policy model is used across entire system and different programs lie under it. For example, file system and a web-server often have different disk access models and therefore have no benefit from the one resource policy used for both of them. [3]

Microkernels are more popular today; they have pushed some of policy decisions into user-level processes. Still, many OS-level policies apply when particular program is in need of resources, therefore applications experience a lot of protection domain crossings, which makes efficiency decrease. These crossing are usually expensive, as they involve entering supervisor mode and changing address spaces [4, 5, 6].

An operating system also needs to be flexibly in terms of its own resources (processes, address space, memory mappings etc). Especially, if we talk about moving some resource management tasks to user-level, operating system should have more flexible and secured mechanisms, and of course overall performance should be taken into account. It is always important to create a stable border between speed and security in such an important issue as operating system kernel development.

Designing a model of exokernel can lead to more efficient operating system. An Aegis project provides direct access to hardware for user-level applications, defining policy decisions to applications, providing secure programmable interface for them. Aegis is an exokernel that provides minimal resource allocation, arbitration services, and a programmable interface [2]. Many early operating systems papers

discussed the need for extendable kernels Lampson's description of the CAL-TSS [18] and Brinch Hansen's microkernel paper [7] are two classic rationales. Hydra was the most ambitious system to have the separation of kernel policy and mechanism as one of its central tenets [8].

The design effort for the Aegis exokernel was focused on the minimal set of abstractions that will enable applications to share the available physical hardware. The Aegis system provides allocation primitives for all of these hardware resources and issue capabilities that authorize their use, therefore making almost no line between hardware and applications and placing an operating system as a third party resource manager, not a resource provider. The exokernel operates a virtual machine that permits applications to safely download code into the supervisor mode execution environment. The whole idea is to create a really thin layer between a demand and recourse instead of commonly used today policies assigning. It is a plus for performance, but a minus for security.

It is important to create some kind of an abstraction level for operating system resources, since hardware resources are mostly going to be controlled by applications themselves. The kernel cache mechanism does exactly that. The cache kernel acts as a HAL (hardware abstraction level), but for OS resources. Kernel calls, address space, memory mappings and processes are loaded into cache before executing or using, and the resulting object is stored back to cache. Each of these objects is viewed as being implemented outside of the cache kernel, just as data in a conventional cache has a backing memory area where it "really" resides [2]. Kernel cache model was tested on shared memory systems and networking multi-computers.

Conclusion

Modern operating system is a complex program, so there is usually always some space to work on and make things better. Considering complexity and variety of hardware resources, it is important to understand, that more universal approach will always lead to less efficient system. When designing of an OS have certain goals and past experience, new ways and ideas can be implemented, and more efficient system can be built. An Aegis exokernel model is interesting approach, because exokernels never had big and real implementation, but the problem was in finding a niche for this kind of systems. Today resource allocation issue means time, and time means money, therefore new innovative ideas are important to work on. An Aegis system defers policy (set of rules for resource use) decisions to applications, providing secure programmable interface. With direct access to hardware, code inspection, using type-safe languages and inline cross-domain calls makes the whole system very flexible for any kind of resource demands from user-level.

When the security and fault-tolerance issues become important in this development, another approach can be used and combined with exokernel – kernel cache mechanism. It provides a cache to load and unload OS resources. These two ideas can be implemented in a single system, which suppose to become another remarkable step on the way of development a flexible operating system.

References

- [1] Dawson Engler, M. Frans Kaashoek, James O’Toole. The Operating System Kernel as a Secure Programmable Machine, page 2. MIT Laboratory for Computer Science, Cambridge, MA, 02139, USA
- [2] David R. Cheriton, Kenneth J. Duda, A Caching Model of Operating System Kernel Functionality. Computer Science Department, page 2, Standford University.
- [3] Kieran Harty and David R. Cheriton. Apphcation-controlled physical memory using external page-cache management. Proc. of the Fifth Conf. on Architectural Support for Programming languages and Operating Systems, pages 187-199, October 1992.
- [4] David Nagle, Richard Uhlig, Tim Stanley, Stuart Sechrest, Trevor Mudge, and Richard Brown. Design tradeoffs for software-managed TLBs. ~Oth Annual International Symposium on Computer Architecture, pages 27-38, 1993.
- [5] J. Bradley Chen and Brian N. Bershad. The impact of operating system structure on memory system performance. Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, 1993.
- [6] T.E. Anderson, H.M, Levy, B.N. Bershad, and E.D. Lazowska. The interaction of architecture and operating system design. In Prec. of the Fourth Conf. on Architectural Support for Programming Languages and Operating Systems, 1991.
- [7] Per Brinch Hansen. The nucleus of a multiprogramming system. Communications of the ACM, 13(4):238-241, April 1970.
- [8] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: The kernel of a multiprocessing operating system. Communications of the ACM, 17(6):337-345, July 1974.