Exokernel

From OSDev Wiki

Exokernels are an attempt to separate security from abstraction, making non-overrideable parts of the operating system do next to nothing but securely multiplex the hardware. The goal is to avoid forcing any particular abstraction upon applications, instead allowing them to use or implement whatever abstractions are best suited to their task without having to layer them on top of other abstractions which may impose limits or unnecessary overhead. This is done by moving abstractions into untrusted user-space libraries called "library operating systems" (libOSes), which are linked to applications and call the operating system on their behalf.

Kernel Designs

Models

Monolithic Kernel
Microkernel
Hybrid Kernel
Exokernel
Nano/Picokernel
Cache Kernel
Virtualizing Kernel
Megalithic Kernel

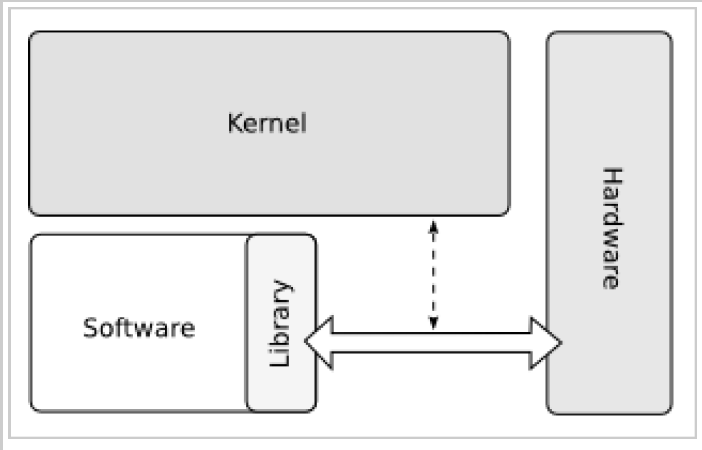
Other Concepts

Modular Kernel
Higher Half Kernel
64-bit Kernel
Bare Bones

Contents

Exokernel Concept

The concept of an exokernel is orthogonal to that of micro- vs. monolithic kernels. It is not important whether the secure multiplexing is performed in privileged kernel code or usermode servers, just that there are no forced abstractions.



Example

Take the typical abstraction of a file. Files, as the application sees them, do not exist on disk. On the disk, there are disk sectors. The operating system abstracts the reality of the disk to create the more convenient illusion of files and a filesystem. Usually, security is also provided at this level. ACLs, Unix-style permissions, etc. are applied to files. Security is combined with the abstraction.

On exokernels, security is provided on the unabstracted hardware level, in this example, to disk sectors. LibOSes provide any desired abstractions on top of this interface. Non-overrideable security is put in the exokernel, and the overrideable abstraction is implemented in a libOS. Security is separated from abstraction.

Advantages

The upside of this is that user-space applications are allowed to implement their own, optimized memory management (by directly accessing e.g. memory tables), file access (by "raw" disk access), etc. This can, for special applications, result in significant performance increases. Engler et al. benchmarked a web server, Cheetah, running on an exokernel being up to eight times faster than the competition. Cheetah, among other things, uses knowledge of HTTP to combine IO requests, knowledge of HTML to appropriately colocate resources on disk, avoids copying data by sending resources to clients directly from the file cache, and caches pregenerated network packets. See Application Performance and Flexibility on Exokernel Systems (http://pdos.csail.mit.edu/papers/exo-sosp97/exo-sosp97.html) .

In addition, by being aware of resource availability, revocation and allocation, it is hoped that applications can make more efficient and intelligent use of hardware resources. For example, a server would know not to make its cache in memory larger than the amount of memory it actually has. In traditional systems, the operating system would fake however much the server had allocated with virtual memory, and the server would carry on, ignorant of any page faults, swapping, and paging.

It is also hoped that exokernels will, in a similar way to microkernels, ease development and testing of new operating system ideas. New scheduling techniques, memory management methods, filesystems, etc. can be developed and tested in libOSes, which are more quickly and easily modified than current operating systems. In The Exokernel Operating System Architecture (http://www.cs.biu.ac.il/~wiseman/2os/microkernels/exokernel.pdf) , Engler recalls the story of an undergraduate who was able to develop and test a new page table structure on an exokernel in a week while the designers had only been able to simulate it.

Disadvantages

Exokernel technology is still not thoroughly researched, so surprises might lurk everywhere. Moreover, the added flexibility for user-space programs also means reduced consistency. While it would be theoretically possible to provide libOSes that enable e.g. MacOS, Windows, and Linux applications to run simultaneously on the same system, that would also mean different look & feels for each of them. In addition, different libOSes may have varying levels of compatibility and interoperability with each other.

It can be difficult to design exokernel interfaces. The designer must develop adequate and appropriate interfaces to low level hardware and delicately balance power and minimalism vs. enough protection. Engler et al. remark that their exokernel interfaces went through many revisions.

The ease of creation and mixing of libOSes could lead to code messes that would be a nightmare for maintenance coders and system administrators. Maintenance coders would have to deal with not only the application code, but any overriden abstractions or new implementations.

Information that could otherwise be useful to a kernel can be lost if the related abstractions are implemented in libOSes rather than the kernel.

Exokernel Derivates

While Monolithic Kernel and Microkernel are rather well-defined terms, the advocates of exokernel-like technology have coined many different terms - nanokernel, picokernel, cache kernel, virtualizing kernel etc. Most of these are relatively minor variations of each other.

Nanokernel/Picokernel

Nanokernels and picokernels are usually small kernels considered by their creators to be even smaller than microkernels. Examples include: Adeos (http://home.gna.org/adeos/) , KeyKOS (http://www.cis.upenn.edu/~KeyKOS/NanoKernel/NanoKernel.html) , and LSE/OS (http://lseos.sourceforge.net/) . Another very famous example is the symbian EKA2 kernel. This nanokernel implements drivers inside the kernel making it not fully a microkernel.

Cache Kernel

The Stanford cache kernel (http://www-dsg.stanford.edu/papers/cachekernel/main.html) caches kernel objects, like address spaces and threads, and allows usermode “application kernels” to manage them, loading and unloading them as needed. Application kernels manage their threads' page faults, exceptions, etc., and the cache kernel allows several of these application kernels to coexist in a single system.

Virtualizing Kernel

Virtualizing kernels are usually designed to allow multiple operating systems to run on a single computer, by allowing free execution of unprivileged instructions and trapping and simulating privileged instructions. Adeos, while it calls itself a nanokernel, is similar in concept to virtualizing kernels. Unlike exokernels, virtualizing kernels attempt to be as transparent as possible, to avoid requiring many modifications, if any, to the hosted operating systems.

Categories: Kernel | OS theory
