The Exokernel Approach to Operating System Extensibility

Dawson R. Engler M. Frans Kaashoek James W. O'Toole Jr.

MIT Laboratory for Computer Science

To provide modularity and performance, operating system (OS) kernels should have only minimal embedded functionality. Today's operating systems are large, inefficient, and most importantly, inflexible. In our view, most OS performance and flexibility problems can be eliminated simply by pushing the OS interface lower. To achieve this goal, we have defined a new OS structure, *exokernel*, that safely exports only those resources defined by the underlying hardware. Our goal is to put abstractions traditionally implemented by the kernel out into user-space, where user-level libraries or servers abstract the exposed hardware resources.[1]

By moving management of resources out into user-space, an exokernel allows aggressive customization. We see three reasons for customization. The first is to tailor policies to given hardware configurations, allowing applications to advance with the hardware. The second is to tailor policies to given applications. For example, closer integration with virtual memory management can enable distributed shared memory systems and garbage collectors to run an order of magnitude faster. The third is to improve performance by reducing the overhead of interrupts, exceptions, context switches, and cache misses, which are the primary sources of overhead in today's OSs.

The exokernel's sole function is to allocate, deallocate, and multiplex physical resources in a secure way. Resource access is controlled through 64-bit capabilities. The resources exported by the exokernel include physical memory (divided into pages), the CPU (divided into time-slices), the TLB, context-identifiers, and disk memory (divided into blocks).

The exokernel performs two non-physical functions: exception forwarding and upcalls. The exokernel hands exceptions directly to applications, enabling efficient user-level exception handling. Upcalls implement synchronous cross-domain transfer of control. Operationally, an upcall transfers the program counter in one domain to an agreed upon value in another, and installs the called domain's exception context. This mechanism is light-weight (our unoptimized implementation takes 20 instructions on a MIPS R3000) and lets applications build their own IPC semantics (i.e., a client can trust a server to save and restore registers).

The exokernel can be *programmed* by down-loading application code into the kernel. This code is made safe by bounding wild jumps and memory operations through sandboxing. Execution time can be bounded through either code augmentation or interrupts. The ability to download code allows applications to perform operations not possible in user-space. For example, this code can access unmapped physical memory, which is important for handling TLB-miss exceptions. Additionally, by allowing user code and state to reside behind the supervisor barrier, the frequency of user/supervisor kernel crossings can be reduced.

The exokernel's low-level interface allows flexible user-level implementations of traditionally rigidly defined OS services. For instance, since the exokernel does not enforce a particular page-table structure, applications can aggressively specialize them. Since the address space of a "protected object" may contain just a few pages, it can be mapped using a small linear page-table, while a single 64-bit address-space subsystem may use inverted page-tables.

To summarize, the advantages of an exokernel-OS are: (1) ambitious systems can be built without fighting inappropriate OS abstractions by accessing physical resources directly, and (2) multiple implementations of OS abstractions can co-exist, allowing them to be less complex and more efficient through specialization to a given domain. Furthermore, as these libraries can reside in the same protection domain as an application, cross-domain calls may be avoided.

No other OS architecture allows this degree of flexibility in the definition and implementation of fundamental OS services.

[1] See http://www.pdos.lcs.mit.edu or "Aegis: a secure programmable exokernel" in the *Proc. of the Sixth European SIGOPS workshop*, Sept. 1994, Wadern, Germany., for more information.