OP1: Exokernel in Disguise

Introduction

Arrakis is a kernel architecture proposed in 2013 [1] that tries to address the same problem that Exokernel [2]: an OS kernel that mediates the hardware is a bottleneck to performance.

Common aspects

In the model of Arrakis and Exokernel's one, applications can directly access the hardware. The kernel is responsible only for providing secure access to devices. Neither provides abstractions on top of the hardware. Arrakis is not involved in data plane operations at all, while Exokernel is for some devices.

Arrakis, like the Exokernel, relegates the OS to the application space (in the application container in the case of Arrakis). In a monolithic or a micro-kernel architecture, the kernel implements a set of hardware abstractions (e.g. the virtual memory). In the case of Arrakis and Exokernel, library operating systems implement them. These libraries are executed in the same space as the application itself so developers can use the library that implements the abstraction that matches best their application's needs.

Divergences

Arrakis and Exokernel differ in the way they provide control plane services. Arrakis provides an abstraction that is a protection domain with an interface to the kernel, called an *application container*. Through this interface, the kernel provides hardware resources allocation to the container and also a mechanism for memory sharing between containers and for directed context switching. It is possible to create multiple protected address spaces inside one container, so implementing a fine-grained sandboxing (e.g. for a web browser) becomes possible. Exokernel, on the other hand, does not provide such an abstraction. It relies on secure binding mechanisms, which are mechanisms that decouple authorization from the use of a resource. The kernel is then capable of managing capabilities and allocation without standing between the application and the resource at use time, letting the application having direct access to the hardware once the binding is effective.

Exokernel introduces ASH (i.e. Application-specific Safe Handler), a mechanism through which applications can load code into the kernel, while Arrakis does not provide something equivalent. Exokernel uses ASH, for example, for incoming packets demultiplexing: the application can provide a piece of code to execute when a packet arrives (e.g., to copy the packet in the application's virtual memory space). To implement the same feature, Arrakis proposes instead to rely on virtualizable hardware. One physical device can offer one virtual device for each application. An application then can configure its virtual NIC to put incoming packets where it needs.

Arrakis and Exokernel manage the revocation differently. With Arrakis, multiple applications can use the same physical device by having multiple virtual ones that the kernel can delete if needed. Exokernel needs to break the secure binding either with application collaboration or by force.

Conclusion

Arrakis and Exokernel are two architectures that apply the "end-to-end" principle to kernels, stating that applications know better their needs and therefore need direct access to hardware. They achieve it by reducing the kernel responsibilities and removing abstractions from the kernel to provide unmediated hardware to applications. Nevertheless, they implement it differently.

[1] Simon Peter, Thomas Anderson, "Arrakis: The Operating System as Control Plane", ;login:, August 2013, Vol.38, No.4

[2] D. R . Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An Operating System Architecture for Application-Level Resource Management," Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995, pp. 251-266.