OP1: Exokernel in Disguise

Arrakis [1] is a proposed OS design that is very similar to the exokernel [2]. Arrakis and the exokernel start from different but similar premises, and arrive at a similar solution. The exokernel started from the idea that monolithic OS design locks applications into a fixed set of abstractions. These might not be the right fit for every application. The exokernel's goal is to provide low-level abstractions, not much higher than a hardware interface. They can be implemented efficiently, with flexibility for implementing higher abstractions on top. Arrakis starts with the premise that IO hardware is getting faster while CPUs are not, resulting in the overhead of having the kernel manage IO resources increasing. Their solution is to eliminate the kernel as a middleman wherever possible by leveraging hardware that can provide multiple virtual interfaces. The kernel would then only be involved in allocation but not in access to these interfaces, eliminating frequent context switches. While Arrakis and the exokernel have different motivations, the solution is the same. They both provide applications with the lowest level of hardware access possible, taken to its extreme by Arrakis by providing secure direct hardware access.

The principle of separation of control plane and data plane is present in both systems. The exokernel achieves this through its 'secure bindings', a protection mechanism that separates authorization from use of a resource. For example, whenever the exokernel allocates a memory region to a process, it hands this process a capability corresponding to that region. The application can present this capability to prove it is allowed to access the memory region. The process can share memory with other processes by sharing the capability. When a process presents such a capability, the kernel will install an entry in the TLB, ensuring that subsequent accesses to the memory region do not require kernel intervention. This idea is pervasive in Arrakis, although the use of capabilities is less pronounced there. Similarly to memory management in the exokernel, Arrakis assigns processes virtual interfaces to hardware but is otherwise not involved in access to these interfaces. Arrakis *does* use capabilities for inter-process file sharing. Applications can request access to files owned by others. If allowed, they receive a capability to access the file with using an indirection through the kernel. To summarize, Arrakis provides separation of the control and data plane by allocating virtual hardware interfaces to processes, while the exokernel does it through its secure bindings.

In the exokernel, the revocation/abort protocol allows applications to coordinate with the kernel when resources have to be revoked. Arrakis does not appear to offer this and it is unclear from the paper how the system will react to something like an out-of-memory event.

In conclusion, at its core Arrakis can be considered as an exokernel that offloads most of its tasks like protection onto the hardware. The benefits they both provide are extremely similar.

References

[1] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The operating system is the control plane. In *Proc. OSDI*, 2014.

[2] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: An operating system architecture for application-level resource management. In *SOSP*, pages 251–266, 1995.