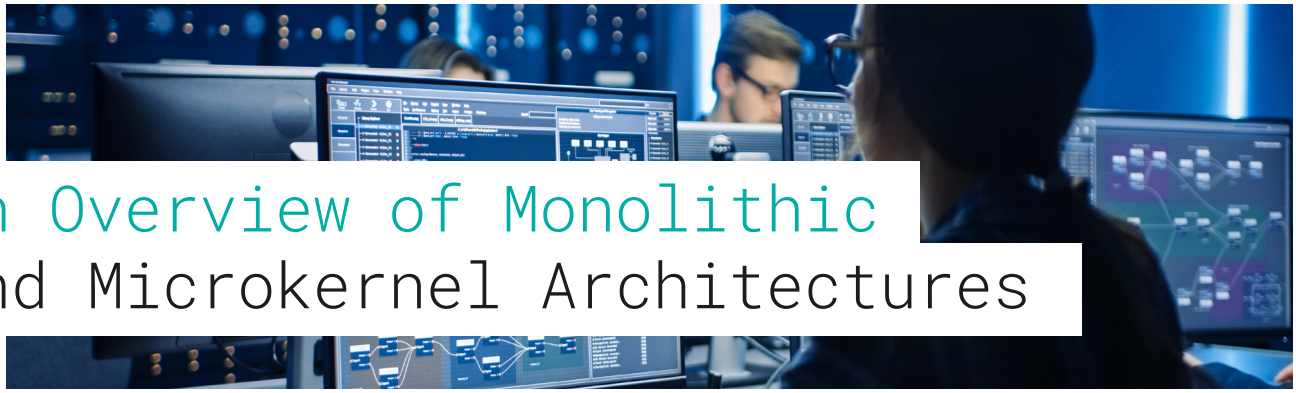

# An Overview of Monolithic and Microkernel Architectures

This document provides an overview of two kernel types: the microkernel and the monolithic kernel.[1] While the focus here is on kernels, developers and users experience kernels to a large degree through their interactions with an operating system. Questions such as, "Which kernel is best for my team or project?" or "Should I really move our development from one kernel type to another?" tend to be answered in the context of an operating system, investments in existing IP on a given OS, and the overhead associated with switching from one OS system to another. Linux is perhaps the best-known example of an OS with a monolithic kernel, and our own VxWorks® also features a monolithic kernel. At Wind River®, discussions about the merit of one kernel versus the other come up almost exclusively in conversations with our automotive customers who are familiar with the microkernel-based QNX Neutrino Real-Time Operating System.

Kernels are typically described as either a process or program at the core of an OS that manages the operation of a system's hardware and software. In a monolithic kernel, the entire OS runs in a single program in kernel mode, i.e., the kernel services and other OS functions such as device drivers, protocol stacks, and file systems are executing in the same address space. In a microkernel implementation, only the minimal number of kernel services run in kernel mode, while all other OS functions execute in user modes and different address spaces. Each approach comes with benefits and drawbacks. With each successive generation, kernels of either type are refined to address the concerns associated with each approach.

The primary advantage of a monolithic kernel is faster processing. Because all elements of the kernel, including the scheduler, file system, memory management, networking stacks, device drivers, and so on, are in the same address space, message passing between address spaces is not required. Known disadvantages of monolithic kernels are that they are larger than microkernel alternatives and historically have required a recompile to add new functionality or perform maintenance updates. Most monolithic kernels today are not completely monolithic — they can dynamically load kernel modules, meaning functionality can be added to the kernel without having to recompile the whole binary.

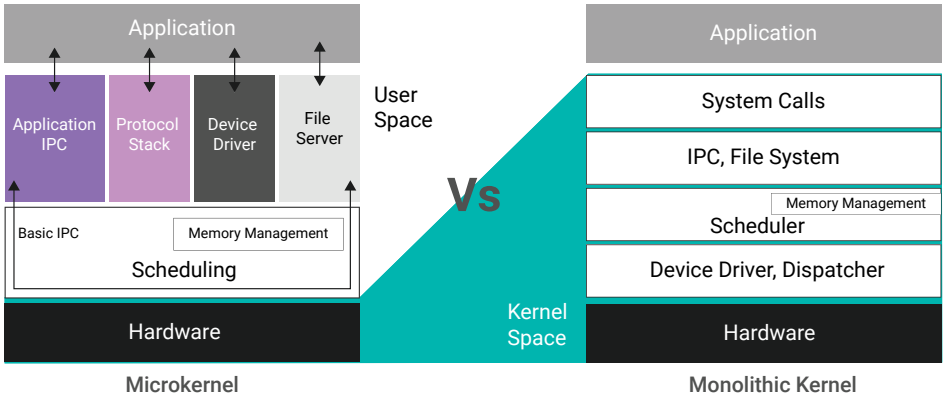


*Figure 1. A comparison of microkernel and monolithic kernel architectures*

1 *There are a minimum of three types of kernel: monolithic, micro, and hybrid (TechTarget). Other sources also describe exokernels and nanokernels.*

The primary advantage of a microkernel architecture is it size, with most microkernels capable of fitting in a given system's L1 cache. This is accomplished by limiting the kernel to only the most important components, such as inter-process communication, basic scheduling, and basic memory management. To support these bare-bones basics, OS capabilities require communication via message passing to other OS capabilities running in user space. This communication from kernel to user space involves an overhead, resulting in slower processing.

## WHY A MONOLITHIC KERNEL?

There are numerous reasons to use a monolithic kernel:

1. Monolithic kernels offer higher performance by reducing layers of abstraction. Communication between kernel components is therefore more efficient, because they are all executed in a single address space.
2. In a monolithic kernel, a service can be obtained through a single system call rather than needing to exchange IPC messages between processes to obtain the service.
3. The design of a monolithic kernel in simpler because the kernel components are all totally integrated. The kernel doesn't have to account for an infinite number of configurations or dependencies that may or may not be present in a microkernel.
4. There is less overhead for context switches because the system calls in a monolithic kernel are all made in the same address space.
5. Device drivers are more responsive and easier to manage because they are integrated into the kernel and operate in the same address space.
6. It is easier to implement, maintain, and upgrade the kernel because all the components are compiled in a single executable.
7. For the same functionality, a monolithic kernel requires less code to implement and is less expensive to certify.

## WHY SHOULD I CARE?

In most scenarios, the design and architecture of an OS used by a software development team is far less important than whether the OS does the job required. If the OS offers the features needed and is reliable, proven, and provides the performance, determinism, and other characteristics needed to fulfill the requirements of the end product, then the way it works under the hood is not a factor likely to impact the project outcome.

For development teams with existing IP developed for an OS with either kernel type, the architectural considerations and trade-offs of monolithic vs. microkernel are secondary to the effort required to migrate between those OSes. Rewriting code to map to a different set of APIs is nontrivial, and managing those changes through an abstraction layer would introduce overhead.

Teams wishing to shift to Wind River while preserving their investment in IP based on an OS with a microkernel architecture should consider Wind River Helix™ Virtualization Platform. Helix Platform can host such an OS.

Also for development teams that prefer a microkernel architecture, the Wind River hypervisor is a Type 1 hypervisor with a messaging microkernel that uses either synchronous or asynchronous communications between systems, events, and beyond to provide deterministic application performance.

WNDRVR