that communicate directly with the virtualization layer. The technique was first used in Denali, a virtual machine monitor that hosts the specially written Ilwaco guest operating system. Xen went one step further by running paravirtualized versions of commodity operating systems.*

Paravirtualizing an operating system involves rewriting all of its code that is incompatible with the paravirtualized architecture. Performance improves because the changes are made in advance, by developers, rather than at runtime. To demonstrate the power of paravirtualization, the Xen team first required an operating system that they could change. Fortunately, Linux was available, open source, and widely used. Only 2,995 lines in the Linux kernel were modified or added to make it run on Xen: this represents less than 2% of the x86 Linux codebase. With paravirtualization (as with virtualization), all of the existing user applications can continue to be used without modification, so the overall modifications are not too invasive.

To achieve paravirtualization, you must either write the operating system yourself (the Denali approach), modify an existing open source operating system (such as Linux or BSD), or convince the developers of a proprietary operating system that paravirtualizing their code is worthwhile. The original research version of Xen achieved impressive, near-native performance on a number of benchmarks running on the modified version of Linux. This performance, and the fact that Xen was released as open source software, has brought us to a point where proprietary operating system developers are paravirtualizing parts of their code, so that they will run more efficiently on hypervisors such as Xen. Even more encouragingly, the paravirtual operations developed for Xen and other hypervisors have been standardized in the latest version of the Linux kernel. By incorporating Xen (and other hypervisor) support in the standard kernel, the uptake of virtualization becomes even easier.

How does paravirtualization work? The full details are too involved to cover here, but the "Further Reading" section at the end of this chapter includes papers that cover the techniques in depth. Here, we'll look at two examples of paravirtualization: for virtual memory and for virtual devices.

The first step in paravirtualizing an operating system is to make it aware that it is not the most privileged software running on the computer; that distinction is awarded to the hypervisor. Most processors have at least two modes: *supervisor* and *user* mode. Normally, the operating system kernel would run in supervisor mode, but this is reserved for Xen, so it must be modified to run in user mode.† However, when running in user mode, several operations are illegal.

* Of course, it could be argued that VM/370—IBM's operating system from the 1960s and the progenitor of virtualization—was the first paravirtualized operating system. However, since IBM designed the instruction set, operating system, and virtual machine monitor, this approach faced different challenges to modern paravirtualization.

† The x86 architecture has four privilege levels, or *rings*, with 0 being the most privileged and 3 the least privileged. In its 32-bit incarnation, Xen ran in ring 0, paravirtualized kernels ran in ring 1, and user applications ran, as normal, in ring 3. However, on the 64-bit version, paravirtualized kernels run in ring 3, due to a difference in the memory segmentation hardware.