

The Wayback Machine - <https://web.archive.org/web/20080203122949/http://tldp.org:80/HOWTO/Module-H...>

Linux Loadable Kernel Module HOWTO

[Prev](#)[Next](#)

2. Introduction to Linux Loadable Kernel Modules

If you want to add code to a Linux kernel, the most basic way to do that is to add some source files to the kernel source tree and recompile the kernel. In fact, the kernel configuration process consists mainly of choosing which files to include in the kernel to be compiled.

But you can also add code to the Linux kernel while it is running. A chunk of code that you add in this way is called a loadable kernel module. These modules can do lots of things, but they typically are one of three things: 1) device drivers; 2) filesystem drivers; 3) system calls. The kernel isolates certain functions, including these, especially well so they don't have to be intricately wired into the rest of the kernel.

2.1. Terminology

Loadable kernel modules are often called just kernel modules or just modules, but those are rather misleading terms because there are lots of kinds of modules in the world and various pieces built into the base kernel can easily be called modules. We use the term loadable kernel module or LKM for the particular kinds of modules this HOWTO is about.

Some people think of LKMs as outside of the kernel. They speak of LKMs communicating with the kernel. This is a mistake; LKMs (when loaded) are very much part of the kernel. The correct term for the part of the kernel that is bound into the image that you boot, i.e. all of the kernel *except* the LKMs, is "base kernel." LKMs communicate with the base kernel.

In some other operating systems, the equivalent of a Linux LKM is called a "kernel extension."

Now what is "Linux"? Well, first of all, the name is used for two entirely different things, and only one of them is really relevant here:

1. The kernel and related items distributed as a package by Linus Torvalds.
2. A class of operating systems that traditionally are based on the Linux kernel.

Only the first of these is really useful in discussing LKMs. But even choosing this definition, people are often confused when it comes to LKMs. Is an LKM part of Linux or not? Though an LKM is always part of the kernel, it is part of Linux if it is distributed in the Linux kernel package, and not otherwise. Thus, if you have loaded into your kernel a device driver LKM that came with your device, you can't, strictly speaking, say that your kernel is Linux. Rather, it's a slight extension of Linux. As you might expect, it is commonplace to use the name "Linux" approximately -- Lots of variations on Linux are in use and are widely distributed, and referred to as "Linux." In this document, though, we will stick to the strictest definition in the interest of clarity.

2.2. History of Loadable Kernel Modules

LKMs did not exist in Linux in the beginning. Anything we use an LKM for today was built into the base kernel at kernel build time instead. LKMs have been around at least since Linux 1.2 (1995).

Device drivers and such were always quite modular, though. When LKMs were invented, only a small amount of work was needed on these modules to make them buildable as LKMs. However, it had to be done on each and

every one, so it took some time. Since about 2000, virtually everything that makes sense as an LKM has at least had the option of being an LKM.

2.3. The Case For Loadable Kernel Modules

You often have a choice between putting a module into the kernel by loading it as an LKM or binding it into the base kernel. LKMs have a lot of advantages over binding into the base kernel and I recommend them wherever possible.

One advantage is that you don't have to rebuild your kernel as often. This saves you time and spares you the possibility of introducing an error in rebuilding and reinstalling the base kernel. Once you have a working base kernel, it is good to leave it untouched as long as possible.

Another advantage is that LKMs help you diagnose system problems. A bug in a device driver which is bound into the kernel can stop your system from booting at all. And it can be really hard to tell which part of the base kernel caused the trouble. If the same device driver is an LKM, though, the base kernel is up and running before the device driver even gets loaded. If your system dies after the base kernel is up and running, it's an easy matter to track the problem down to the trouble-making device driver and just not load that device driver until you fix the problem.

LKMs can save you memory, because you have to have them loaded only when you're actually using them. All parts of the base kernel stay loaded all the time. And in real storage, not just virtual storage.

LKMs are much faster to maintain and debug. What would require a full reboot to do with a filesystem driver built into the kernel, you can do with a few quick commands with LKMs. You can try out different parameters or even change the code repeatedly in rapid succession, without waiting for a boot.

LKMs are not slower, by the way, than base kernel modules. Calling either one is simply a branch to the memory location where it resides. [\[1\]](#)

Sometimes you *have* to build something into the base kernel instead of making it an LKM. Anything that is necessary to get the system up far enough to load LKMs must obviously be built into the base kernel. For example, the driver for the disk drive that contains the root filesystem must be built into the base kernel.

2.4. What LKMs Can't Do

There is a tendency to think of LKMs like user space programs. They do share a lot of their properties, but LKMs are definitely not user space programs. They are part of the kernel. As such, they have free run of the system and can easily crash it.

2.5. What LKMs Are Used For

There are six main things LKMs are used for:

- **Device drivers.** A device driver is designed for a specific piece of hardware. The kernel uses it to communicate with that piece of hardware without having to know any details of how the hardware works. For example, there is a device driver for ATA disk drives. There is one for NE2000 compatible Ethernet cards. To use any device, the kernel must contain a device driver for it.
- **Filesystem drivers.** A filesystem driver interprets the contents of a filesystem (which is typically the contents of a disk drive) as files and directories and such. There are lots of different ways of storing files and directories and such on disk drives, on network servers, and in other ways. For each way, you need a

filesystem driver. For example, there's a filesystem driver for the ext2 filesystem type used almost universally on Linux disk drives. There is one for the MS-DOS filesystem too, and one for NFS.

- System calls. User space programs use system calls to get services from the kernel. For example, there are system calls to read a file, to create a new process, and to shut down the system. Most system calls are integral to the system and very standard, so are always built into the base kernel (no LKM option). But you can invent a system call of your own and install it as an LKM. Or you can decide you don't like the way Linux does something and override an existing system call with an LKM of your own.
- Network drivers. A network driver interprets a network protocol. It feeds and consumes data streams at various layers of the kernel's networking function. For example, if you want an IPX link in your network, you would use the IPX driver.
- TTY line disciplines. These are essentially augmentations of device drivers for terminal devices.
- Executable interpreters. An executable interpreter loads and runs an executable. Linux is designed to be able to run executables in various formats, and each must have its own executable interpreter.

Notes

[1] For the pedantic, see [Section 10.7](#).

[Prev](#)
Preface

[Home](#)

[Next](#)
Making Loadable Kernel Modules