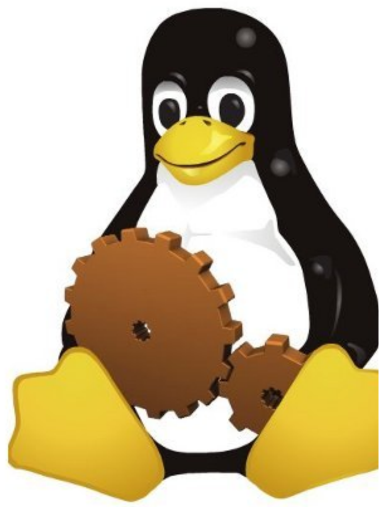
Device Drivers, Part 1: Linux Device Drivers for Your Girl Friend

By **Anil Kumar Pugalia** - November 1, 2010



This series on Linux device drivers aims to present the usually technical topic in a way that is more interesting to a wider cross-section of readers.
"After a week of hard work, we finally got our driver working," were Pugs' first words when he met his girlfriend, Shweta.

"Why? What was your driver up to? Was he sick? And what hard work did you do?" asked Shweta. Confused, Pugs responded, "What are you talking about?"

Now it was Shweta's turn to look puzzled, as she replied, "Why ask me? You tell me — which of your drivers are you talking about?"

When understanding dawned on him, Pugs groaned, "Ah c'mon! Not my car drivers — I am talking about a device driver on my computer."

"I know about car and bus drivers, pilots, and even screwdrivers; but what is this 'device driver'?" queried Shweta, puzzled.

That was all it took to launch Pugs into a passionate explanation of device drivers for the newbie — in particular, Linux devic drivers, which he had been working on for many years.

Of drivers and buses

A driver drives, manages, controls, directs and monitors the entity under its command. What a bus driver does with a bus, a device driver does with a computer device (any piece of hardware connected to a computer) like a mouse, keyboard, monitor, hard disk, Web-camera, clock, and more.

Further, a "pilot" could be a person or even an automatic system monitored by a person (an auto-pilot system in airliners, for example). Similarly, a specific piece of hardware could be controlled by a piece of software (a device driver), or could be controlled by another hardware device, which in turn could be managed by a software device driver. In the latter case, such a controlling device is commonly called a device controller. This, being a device itself, often also needs a driver, which is commonly referred to as a bus driver.

General examples of device controllers include hard disk controllers, display controllers, and audio controllers that in turn manage devices connected to them. More technical examples would be an IDE controller, PCI controller, USB controller, SPI controller, I2C controller, etc. Pictorially, this whole concept can be depicted as in Figure 1.

Device and driver interaction

Figure 1: Device and driver interaction

Device controllers are typically connected to the CPU through their respectively named buses (collection of physical lines) — for example, the PCI bus, the IDE bus, etc. In today's embedded world, we encounter more micro-controllers than CPUs; these are the CPU plus various device controllers built onto a single chip. This effective embedding of device controllers primarily reduces cost and space, making it suitable for embedded systems. In such cases, the buses are integrated into th chip itself. Does this change anything for the drivers, or more generically, on the software front?
The answer is, not much — except that the bus drivers corresponding to the embedded device controllers are now develope under the architecture-specific umbrella.

Drivers have two parts

Bus drivers provide hardware-specific interfaces for the corresponding hardware protocols, and are the bottom-most horizontal software layers of an operating system (OS). Over these sit the actual device drivers. These operate on the underlying devices using the horizontal layer interfaces, and hence are device-specific. However, the whole idea of writing these drivers is to provide an abstraction to the user, and so, at the other "end", these do provide an interface (which varies from OS to OS). In short, a device driver has two parts, which are: a) device-specific, and b) OS-specific. Refer to Figure 2.

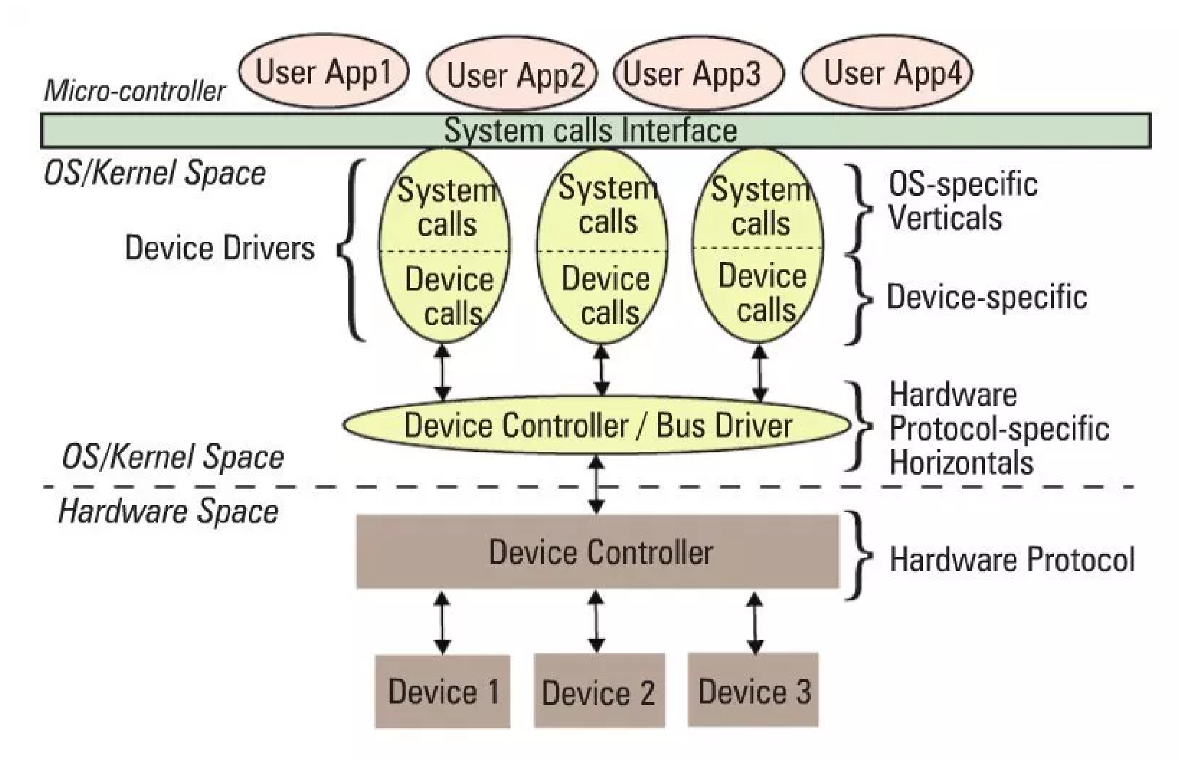


Figure 2: Linux device driver partition

The device-specific portion of a device driver remains the same across all operating systems, and is more about understanding and decoding the device data sheets than software programming. A data sheet for a device is a document with technical details of the device, including its operation, performance, programming, etc. — in short a device user manual. Later, I shall show some examples of decoding data sheets as well. However, the OS-specific portion is the one that is tightly coupled with the OS mechanisms of user interfaces, and thus differentiates a Linux device driver from a Windows device driver and from a MacOS device driver.

Verticals

In Linux, a device driver provides a “system call” interface to the user; this is the boundary line between the so-called kernel space and user-space of Linux, as shown in Figure 2. Figure 3 provides further classification.

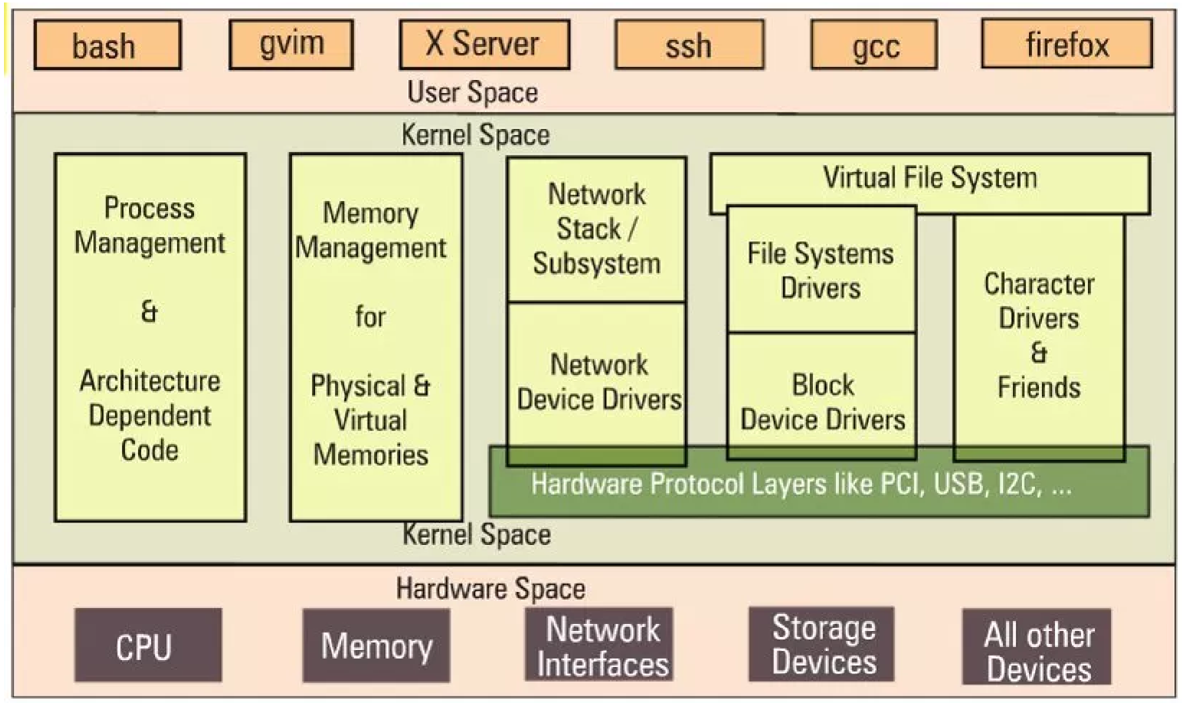


Figure 3: Linux kernel overview

Based on the OS-specific interface of a driver, in Linux, a driver is broadly classified into three verticals:

- Packet-oriented or the network vertical
- Block-oriented or the storage vertical
- Byte-oriented or the character vertical

The CPU vertical and memory vertical, taken together with the other three verticals, give the complete overview of the Linux kernel, like any textbook definition of an OS: “An OS performs 5 management functions: CPU/process, memory, network, storage, device I/O.” Though these two verticals could be classified as device drivers, where CPU and memory are the respective devices, they are treated differently, for many reasons.

These are the core functionalities of any OS, be it a micro-kernel or a monolithic kernel. More often than not, adding code in these areas is mainly a Linux porting effort, which is typically done for a new CPU or architecture. Moreover, the code in these two verticals cannot be loaded or unloaded on the fly, unlike the other three verticals. Henceforth, when we talk about Linux device drivers, we mean to talk only about the latter three verticals in Figure 3.

Let’s get a little deeper into these three verticals. The network vertical consists of two parts: a) the network protocol stack, and b)the network interface card (NIC) device drivers, or simply network device drivers, which could be for Ethernet, Wi-Fi, any other network horizontals. Storage, again, consists of two parts: a) File-system drivers, to decode the various formats different partitions, and b) Block device drivers for various storage (hardware) protocols, i.e., horizontals like IDE, SCSI, MT etc.

With this, you may wonder if that is the only set of devices for which you need drivers (or for which Linux has drivers). Holc on a moment; you certainly need drivers for the whole lot of devices that interface with the system, and Linux does have drivers for them. However, their byte-oriented cessibility puts all of them under the character vertical — this is, in reality, th majority bucket. In fact, because of the vast number of drivers in this vertical, character drivers have been further sub-classified — so you have tty drivers, input drivers, console drivers, frame-buffer drivers, sound drivers, etc. The typical horizontals here would be RS232, PS/2, VGA, I2C, I2S, SPI, etc.

Get real-time push notifications

Multiple-vertical drivers

One final note on the complete picture (placement of all the drivers in the Linux driver ecosystem): the horizontals like USB, PCI, etc, span below multiple verticals. Why is that?

Simple — you already know that you can have a USB Wi-Fi dongle, a USB pen drive, and a USB-to-serial converter — all are USB, but come under three different verticals!

In Linux, bus drivers or the horizontals, are often split into two parts, or even two drivers: a) device controller-specific, and b) an abstraction layer over that for the verticals to interface, commonly called cores. A classic example would be the USB controller drivers ohci, ehci, etc., and the USB abstraction, usbcore.

Summing up

So, to conclude, a device driver is a piece of software that drives a device, though there are so many classifications. In case drives only another piece of software, we call it just a driver. Examples are file-system drivers, usbcore, etc. Hence, all devi drivers are drivers, but all drivers are not device drivers.

“Hey, Pugs, hold on; we’re getting late for class, and you know what kind of trouble we can get into. Let’s continue from her later,” exclaimed Shweta.

Jumping up, Pugs finished his explanation: “Okay. This is the basic theory about device drivers. If you’re interested, later, I can show you the code, and all that we have been doing for the various kinds of drivers.” And they hurried towards their classroom.

Anil Kumar Pugalia

The author is a freelance trainer in Linux internals, Linux device drivers, embedded Linux and related topics. Prior to this, he had worked at Intel and Nvidia. He has been exploring Linux since 1994. A gold medallist from the Indian Institute of Science, Linux and knowledge-sharing are two of his

many passions.