### The role of Device Drivers

Kernel device driver code runs at a high privilege level, as does the rest of the kernel. It has full access to the processor address space and hardware registers. It can handle interrupts and DMA transfers. It can make use of the sophisticated kernel infrastructure for synchronization and memory management.

**no policy in the kernel**. It is the responsibility of user space to set the policy that governs the overall behavior of the system. For example, the loading of kernel modules in response to external events, such as plugging in a new USB device, is the responsibility of the user space program, udev, not the kernel. The kernel just supplies a means of loading a kernel module.

#### Character Devices

Character devices are identified in user space by a special file called a device node. This file name is mapped to a device driver using the major and minor numbers associated with it. Broadly speaking, the major number maps the device node to a particular device driver, and the minor number tells the driver which interface is being accessed.

 In fact, from Linux 2.6 onwards, the major number is 12 bits long, which gives valid numbers from 1 to 4,095, and the minor number is 20 bits, from 0 to 1,048,575.

When you open a character device node, the kernel checks to see whether the major and minor numbers fall into a range registered by a character device driver . If so, it passes the call to the driver, otherwise the open call fails. The device driver can extract the minor number to find out which hardware interface to use.

#### Block Devices

Block devices are also associated with a device node, which also has major and minor numbers.

Although character and block devices are identified using major and minor numbers, they are in different namespaces. A character driver with a major number 4 is in no way related to a block driver with a major number 4.

With block devices, the major number is used to identify the device driver and the minor number is used to identify the partition.

The major number for the MMC block driver is 179 (you can look it up in devices.txt). The minor numbers are used in ranges to identify different physical MMC devices, and the partitions of the storage medium that are on that device. In the case of the MMC driver, the ranges are eight minor numbers per device: the minor numbers from 0 to 7 are for the first device, the numbers from 8 to 15 are for the second, and so on. Within each range, the first minor number represents the entire device as raw sectors, and the others represent up to seven partitions. On eMMC chips, there are two 128 KiB areas of memory reserved for use by a bootloader. These are represented as devices: mmcblk1boot0 and mmcblk1boot1, and they have minor numbers 16 and 24.

SCSI disk driver, known as sd, which is used to control a range of disks that use the SCSI command set, which includes SCSI, SATA, USB mass storage, and universal flash storage (UFS). It has the major number 8 and ranges of 16 minor numbers per interface (or disk). The minor numbers from 0 to 15 are for the first interface with device nodes named sda up to sda15, the numbers from 16 to 31 are for the second disk with device nodes sdb up to sdb15, and so on. This continues up to the 16 disk from 240 to 255 with the node name sdp.

Both the MMC and SCSI block drivers expect to find a partiton table at the start of the disk. The partition table is created using utilities such as fdisk, sfidsk, or parted.

A user space program can open and interact with a block device directly via the device node. This is not a common thing to do, though, and is usually only done to perform administrative operations such as creating partitions, formatting a partition with a filesystem, and mounting. Once the filesystem is mounted, you interact with the block device indirectly through the files in that filesystem.

#### Network Devices

Network devices are not accessed through device nodes, and they do not have major and minor numbers. Instead, a network device is allocated a name by the kernel, based on a string and an instance number. Here is an example of the way a network driver registers an interface.

my\_netdev = alloc\_netdev(0, "net%d", NET\_NAME\_UNKNOWN, netdev\_setup);

ret = register\_netdev(my\_netdev);

This creates a network device named net0 the first time it is called, net1 the second time, and so on. More common names are lo, eth0, and wlan0.

Usually, the network interface name is only used when configuring the network using utilities, such as ip and ifconfig, to establish a network address and route. Thereafter, you interact with the network driver indirectly by opening sockets, and letting the network layer decide how to route them to the right interface. However, it is possible to access network devices directly from user space by creating a socket and using the ioctl commands listed in include/linux/sockios.h.

#### Getting information from sysfs

You can define sysfs in a pedantic way as a representation of kernel objects, attributes, and relationships. A kernel object is a directory, an attribute is a file, and a relationship is a symbolic link from one object to another. From a more practical point of view, since the Linux device driver model represents all devices and drivers as kernel objects, you can see the kernel's view of the system laid out before you by looking in /sys.