Device Management

THE LINUX FOUNDATION

Types of Devices

There are three main types of devices:

- **Character devices** are sequential streams; they mainly implement **open**, **close**, **read**, and **write** functions, e.g. serial and parallel ports (**`/dev/ttyS0`**, **`/dev/lp1`**), sound cards (**`/dev/dsp0`**), etc.
- **Block devices** are randomly accessed only in block-size multiples, and I/O operations are usually cached and deal with mounted filesystems, e.g. hard drive partitions (**`/dev/sda1`**, **`/dev/sdb8`**), CD-ROMs, etc.
- **Network devices** transfer packets of data, not blocks or streams, and usually employ a socket interface; packet reception/transmission functions replace read/write operations, and there are no corresponding filesystem nodes; instead, the interfaces are identified by a name, such as **`eth0`** or **`wlan0`**.

Other Types of Devices

- There are also other types of devices which are classified somewhat differently, according to the type of controller bus they are attached to, such as **SCSI** (**S**mall **C**omputers **S**ystems **I**nterconnect) and **USB** (**U**niversal **S**erial **B**us)
- These devices share an underlying protocol regardless of function
- Besides the driver for the device itself, hard work goes into writing the driver for the controller hardware which may run many devices
- One also has **user-space drivers**, which work completely in user-space, but are given hardware access privileges; these are not as efficient as in-kernel drivers, but are less likely to bring a system down

Device Nodes

- Character and block devices have filesystem entries associated with them
- These **nodes** can be used by user-level programs to communicate with the device, using normal I/O system calls such as **`open()`**, **`close()`**, **`read()`**, and **`write()`**
- A device driver can manage more than one device node

/dev Directory

- Device nodes are normally placed in the **/dev** directory and can be created with:

  ```
  $ sudo mknod [-m mode] /dev/name <type> <major> <minor>
  ```

  e.g.

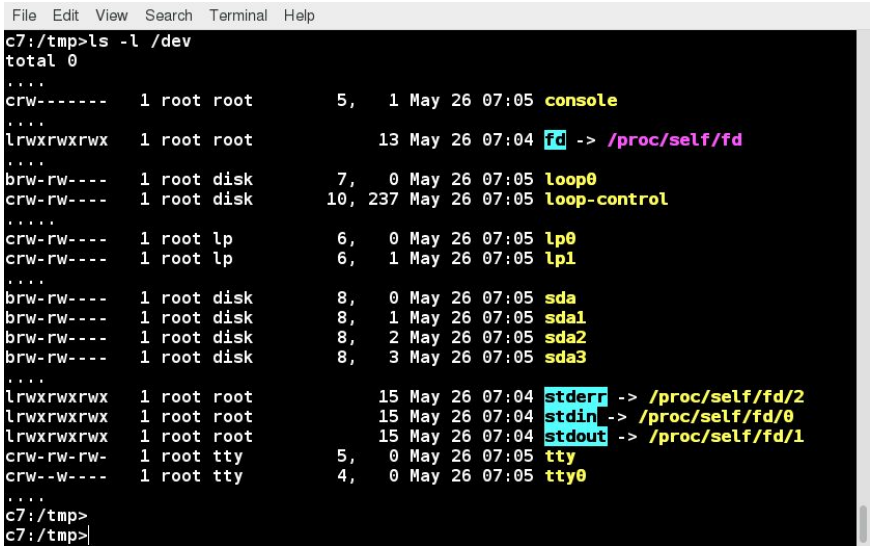
  ```
  $ mknod -m 666 /dev/mycdrv c 254 1
  ```

  or from the **mknod()** system call

- The major number identifies the driver associated with the device; all device nodes of the same type (block or character) with the same major number use the same driver
- The minor number is used only by the device driver to differentiate between the different devices it may control
  - Either different instances of the same kind of device, (such as the first and second sound card, or hard disk partition) or
  - Different modes of operation of a given device (e.g. different density floppy drive media)

/dev Directory (Cont.)

- Device numbers have meaning in user-space as well
- Two POSIX system calls, **`mknod()`** and **`stat()`** return information about major and minor numbers



/dev Directory

Managing Device Nodes

- The methods of managing device nodes became clumsy and difficult as Linux evolved
- The number of device nodes lying in **`/dev`** and its subdirectories reached numbers in the 15,000 - 20,000 range in most installations during the 2.4 kernel series
- Nodes for all kinds of devices which would never be used on most installations were still created by default, as distributors could never be sure exactly which hardware would be needed
- Many developers and system administrators trimmed the list to what was actually needed, especially in embedded configurations, but this was essentially a manual and potentially error-prone task

Managing Device Nodes (Cont.)

- Note that while device nodes are not normal files and do not take up significant space on the filesystem, having huge directories slowed down access to device nodes, especially upon first usage
- Furthermore, exhaustion of available major and minor numbers required a more modern and dynamic approach to the creation and maintenance of device nodes
- Ideally, one would like to register devices by name; however, major and minor numbers cannot be gotten rid of altogether, as POSIX requires them

udev

- The **udev** method creates device nodes on the fly as they are needed; there is no need to maintain a ton of device nodes that will never be used
- The **u** in udev stands for user, and indicates that most of the work of creating, removing, and modifying devices nodes is done in user-space
- udev handles the dynamical generation of device nodes and it evolved to replace earlier mechanisms such as devfs and hotplug
- It supports persistent device naming; names need not depend on the order of device connection or plugging in - such behavior is controlled by specification of udev rules
- udev runs as a **daemon** and monitors a **netlink** socket
    - When new devices are initialized or removed the uevent kernel facility sends a message through the socket which udev receives and takes appropriate action to create or remove device nodes of the right names and properties according to the rules

udev Components

The three components of udev are:

- The **libudev** library which allows access to information about the devices
- The **udevd** daemon that manages the **`/dev`** directory
- The **udevadm** utility for control and diagnostics

Creating and Removing Device Nodes

- As devices are added or removed from the system, working with the hotplug subsystem, udev acts upon notification of events to create and remove device nodes
- The information necessary to create them with the right names, major and minor numbers, permissions, etc., are gathered by examination of information already registered in the **sysfs** pseudo-filesystem (mounted at **`/sys`**) and a set of configuration files
- The main configuration file is **`/etc/udev/udev.conf`**; it contains information such as where to place device nodes, default permissions and ownership, etc.
- By default, rules for device naming are located in the **`/etc/udev/rules.d`** directory
- By reading the **man** page for udev, one can get a lot of specific information about how to set up rules for common situations