Chapter 27 Devices and udev - Notes

27.2 Introduction

Linux uses **udev**, an intelligent apparatus for discovering hardware and peripheral devices both during boot, and later when they are connected to the system. **Device nodes** are created automatically, and then used by applications/operating system subsystems to communicate with + transfer data to/from devices. System administrators can control how **udev** operates and craft special **udev** rules to assure desired behavior results.

27.3 Learning Objectives:

- Explain the role of **device nodes** and how they use **major** and **minor** numbers.
- Understand the need for the **udev** method and list its key components.
- Describe how the **udev** device manager functions.
- Identify **udev** rule files and learn how to create custom rules.

27.4 Device Nodes

Character and block devices have filesystem entries associated with them. Networ k devices in Linux *do not*. These **device nodes** can be used by programs to communicate with devices, using normal I/O system calls such as `open()`, `close()`, `read()`, and `write()`. On other hand, network devices work by transmitting/receiving **packets**, which must be constructed by breaking up streams of data, or reassembled into streams when received.

A device driver may manage multiple device nodes, which are normally placed in `/dev` directory. Can be seen with:

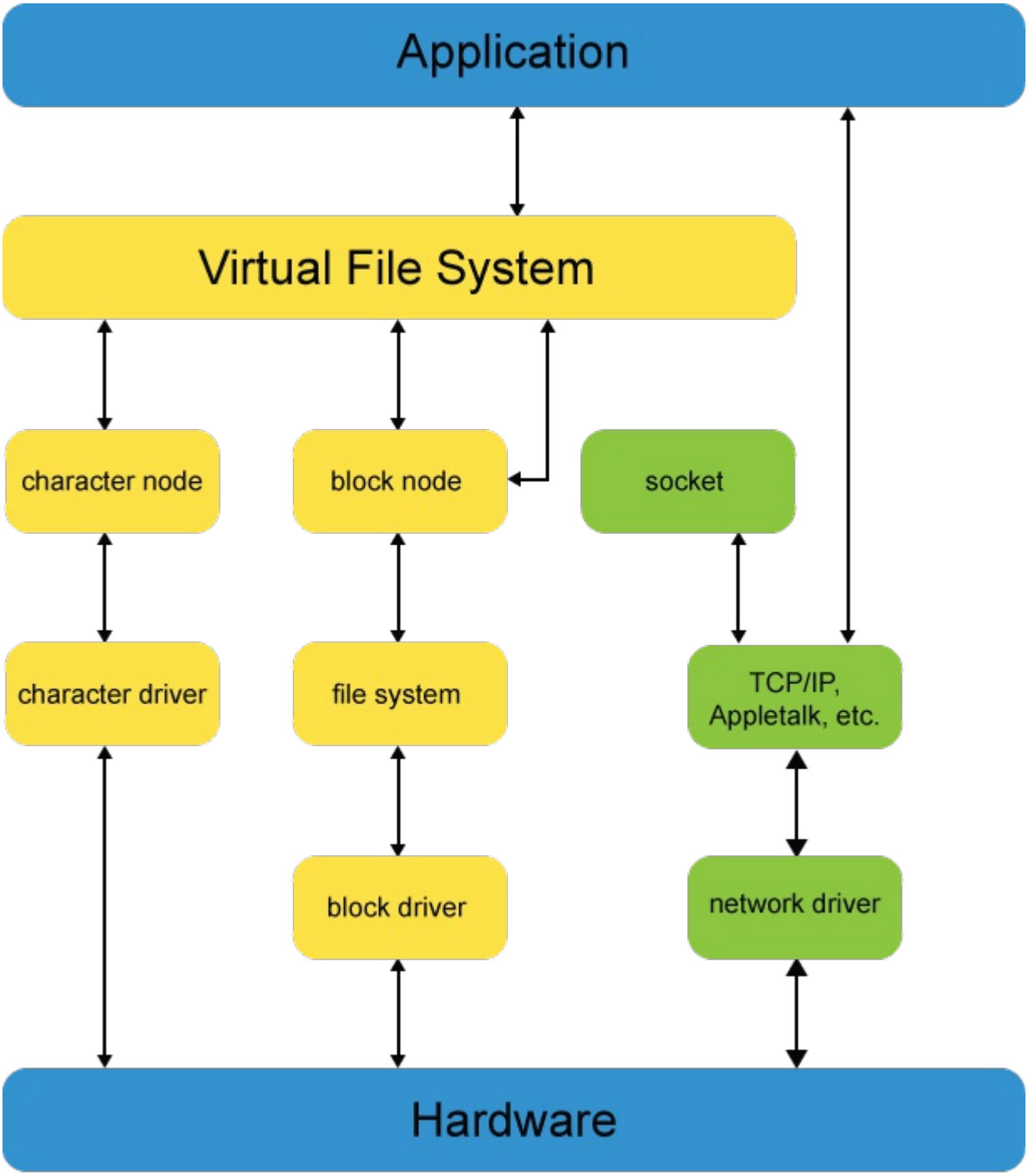
```
$ ls -l /dev
```

Device nodes can be created with:

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

For example:

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



Device nodes

27.5 Major and Minor Numbers

Major and **minor** numbers identify driver associated with device, with driver uniquely reserving a group of numbers. In most cases (but not all), device nodes of the same type (block or character) with the same major number use the same driver.

If you list some device nodes:

```
$ ls -l /dev/sda*
brw-rw---- 1 root disk 8,  0 Dec 29 06:40 /dev/sda
brw-rw---- 1 root disk 8,  1 Dec 29 06:40 /dev/sda1
brw-rw---- 1 root disk 8,  2 Dec 29 06:40 /dev/sda2
.......
```

Major and **minor** numbers appear in same place that file size would when looking at normal file. In above example as **8**, **1**, etc. While normal users will probably never need to refer explicitly to major and minor numbers and will refer to devices by name, system administrators may have to untangle them from time to time if system gets confused about devices, or has some hardware added at runtime.

Minor number used only by device driver to differentiate between different devices it may control, or how they are used. These may either be different instances of same kind of device (such as first and second sound card, or hard disk partition), or different modes of operation of given device (such as different density floppy drive media).

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

27.6 udev

Methods of managing device nodes became clumsy and difficult as Linux evolved. Number of device nodes lying in `/dev` + its subdirectories reached numbers in 15,000 - 20,000 range in most installations during 2.4 kernel version series. Nodes for 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 present on system.

Many developers + system administrators trimmed list to what was actually needed, especially in embedded configurations, but this essentially manual and potentially error-prone task.

Note: while device nodes *not* normal files and don't take up significant space on filesystem, having huge directories slowed down access to device nodes, especially upon first usage. Exhaustion of available major and minor numbers required more modern + dynamic approach to creation/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** standard requires them. (POSIX: acronym for **Portable Operating System Interface**, a family of standards designed to ensure compatibility between different operating systems.)

udev method created device nodes on the fly as needed. 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, modifying nodes is done in user-space.

udev handles dynamical generation of device nodes, evolved to replace earlier mechanisms such as **devfs**, **hotplug**. Supports **persistent device naming**. Names need not depend on order of device connection or plugging in. Such behavior controlled by specification of **udev rules**.

27.7 udev Components

udev runs as **daemon** (either **udevd** or **systemd-udevd**), monitors a **netlink** socket. When new devices initialized/removed, **uevent** kernel facility sends message through the socket, which **udev** receives and takes appropriate action to create/remove device nodes of right names and properties according to the rules.

Three components of **udev**:

1. The **libudev** library which allows access to information about the devices
2. The **udevd** or **systemd-udevd** daemon that manages the `dev` directory
3. The **udevadm** utility for control and diagnostics

Cleanest way to use **udev**: to have a pure system. `dev` directory empty upon initial kernel boot, then populated with device nodes as they are needed. When used this way, must boot using **initramfs** image, which may contain set of preliminary device nodes, as well as **udev** infrastructure.

27.8 udev and Hotplug

As devices added/removed from system, working with the hotplug subsystem, **udev** acts upon notification of events to create/remove device nodes. Information necessary to create them with the right names, major and minor numbers, permissions,

etc., gathered by examination of information already registered in **sysfs** pseudo-filesystem (mounted at `/sys`) and a set of configuration files.

Main configuration file: `/etc/udev/udev.conf`. Contains information such as where to place device nodes, default permissions and ownership, etc. By default, rules for device naming located in `/etc/udev/rules.d` and `/usr/lib/udev/rules.d` directories. BY reading **man** page for **udev**, can get lot of specific information about how to set up rules for common situations.

27.9 The udev Device Manager

When **udev** receives message from kernel about devices being added/removed, it parses the rule-setting files in `/etc/udev/rules.d/*.rules` and `/usr/lib/udev/rules.d/*.rules` to see if there are any rules relevant to device being added/removed.

It then takes appropriate actions including:

- Device node naming
- Device node and symbolic links creation
- Setting file permissions and ownership for the device node
- Taking other actions to initialize and make device available.

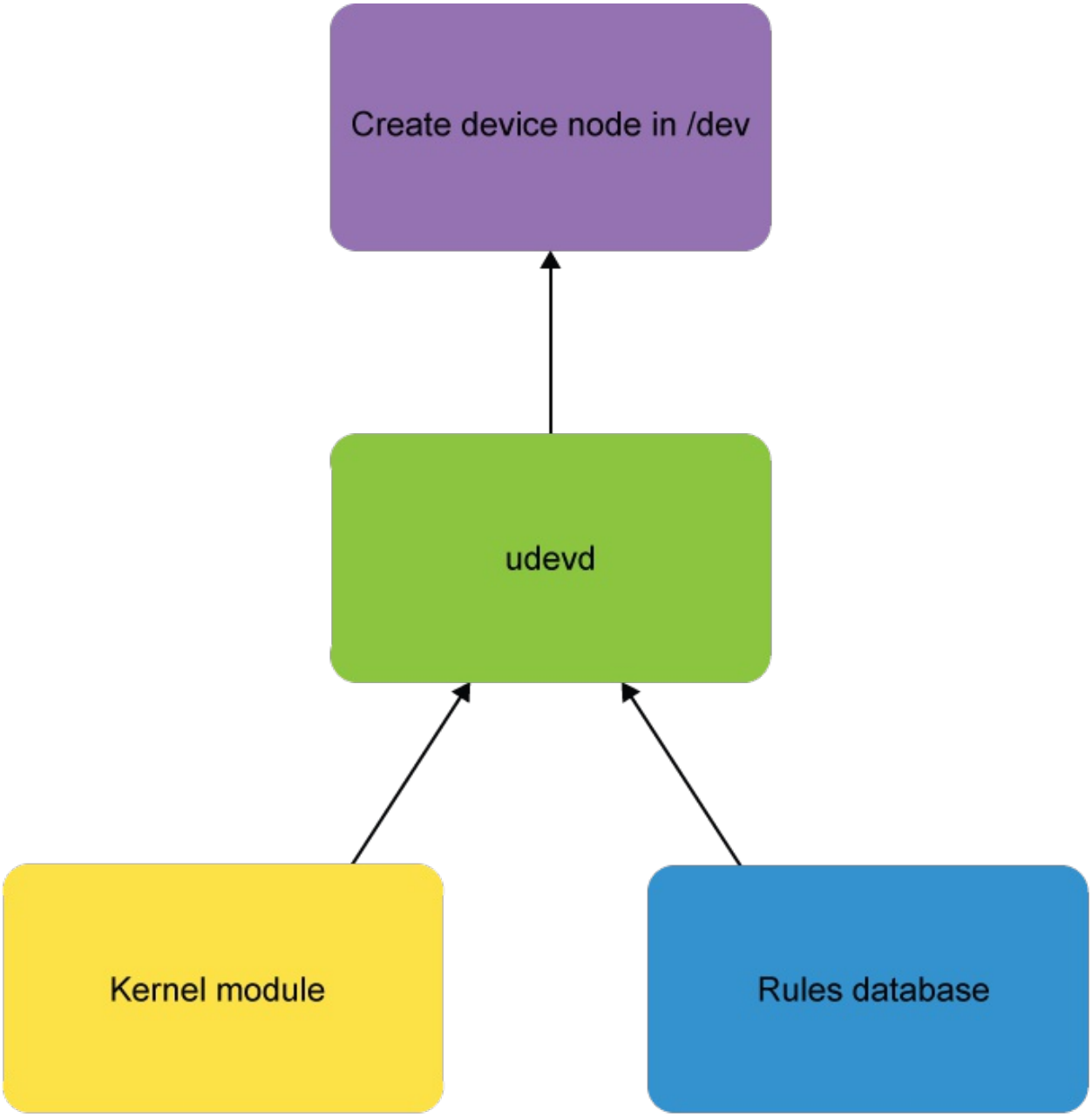
These rules are completely customizable.

27.10 udev Rule Files

udev rules files are located under `/etc/udev/rules.d/<rulename>.rules` with names like:

- `30-usb.rules`
- `90-mycustom.rules`

By default, when **udev** reads the rules files, looks for files that have suffix of `.rules`. If it finds more than one file, reads them one by one, lexicographically, ie. in ascending alphabetical order. Standard rule file name: generally a two digit number followed by descriptive name (for the rules), folowed by `.rules` suffix.



udev Rule

27.11 Creating udev Rules

Format for **udev** rule simple:

```
<match><op>value [, ...] <assignment><op>value [, ...]
```

Two separate files defined on single line:

- First part consists of one or more match pairs denoted by `==`. These try to match device's attributes and/or characteristics to some value
- Second part consists of one or more assignment key-value pairs that assign a value to a name, such as a file name, group assignment, even file permissions etc.

If no matching rule found, uses default device node name and other attributes.

27.12 Some Examples of Rules Files

Example of rules file for **Fitbit** device:

```
$ cat /usr/lib/udev/rules.d/99-fitbit.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", SYMLINK+="fitbit", MODE="0666"
```

Example for creating crash dumps and fast kernel loading with **kdump**/**kexec**:

```
$ cat /usr/lib/udev/rules.d/98-kexec.rules
SUBSYSTEM=="cpu", ACTION=="online", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="cpu", ACTION=="offline", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="add", PROGRAM="/bin/systemctl try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="remove", PROGRAM="/bin/systemctl try-restart kdump.service"
```

Example for kvm virtual machine hypervisor:

```
$ cat /usr/lib/udev/rules.d/80-kvm.rules
KERNEL=="kvm", GROUP="kvm", MODE="0666"

$ cat /usr/lib/udev/rules.d/99-fuse.rules
KERNEL=="fuse", MODE="0666",OWNER="root",GROUP="root"
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter