# The Sysfs Virtual Filesystem

## Exploring the Linux Device Model

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer

# Overview

Roadmap:

- What is *sysfs*?
- What is a *virtual filesystem*?
- What are *attributes*?
- Examples!

# Sysfs

A key component of 2.6 kernels:

- A *virtual filesystem*
- Reflects the kernel’s *device model*
- Tightly coupled with *Device Model API*
- Usually mounted under /texttt/sys

# Sysfs

```
# ls -l /sys
total 0
drwxr-xr-x  2 root root 0 May 28 01:23 block
drwxr-xr-x 14 root root 0 Jan  1  1970 bus
drwxr-xr-x 46 root root 0 Jan  1  1970 class
drwxr-xr-x  4 root root 0 May 28 01:23 dev
drwxr-xr-x  6 root root 0 Jan  1  1970 devices
drwxr-xr-x  2 root root 0 May 28 01:23 firmware
drwxr-xr-x  3 root root 0 May 28 01:23 fs
drwxr-xr-x  5 root root 0 Jan  1  1970 kernel
drwxr-xr-x 62 root root 0 May 28 01:23 module
drwxr-xr-x  2 root root 0 May 27 11:27 power
```

# Sysfs

```
# ls -l /sys/devices/platform
ehci-omap.0    mmci-omap-hs.0     omap-previewer
gpio-keys      mmci-omap-hs.1     omap-resizer.2
i2c_omap.1     musb_hdrc          omap2-nand
i2c_omap.2     omap-iommu.0       omap2_mcspi.1
i2c_omap.3     omap-mcbsp.1       omap2_mcspi.2
leds-gpio      omap-mcbsp.2       omap2_mcspi.3
...
```

# Sysfs

*Virtual filesystem*:

- Look like files to user applications
- No storage on persistent media
- Similar to “ramdisks”, etc.

# Classes, Busses, etc.

There are many “devices” in `/sys`:

- True devices, like `musb_hdrc`
- Synthetic devices, like `power`
- Virtual devices, like `input`
- ??? devices, like modules

# Classes, Busses, etc.

Sysfs is more than devices:

- It’s really a database of *kernel objects*
- `struct kobject`

# Classes, Busses, etc.

Kobjects represent:

- Modules
- Devices
- Interfaces
- Memory

... Everything!

# Classes, Busses, etc.

Sysfs entries are sorted:

- ... by bus type
- ... by object type
- ... by device type
- ... by parent/child relationships
- ...

Redundancies are reduced with symlinks

# Classes, Busses, etc.

```
# ls -l .../i2c_omap.1/i2c-1/subsystem
.../subsystem -> ../../../../bus/i2c
```

# Attributes

A characteristic of the target object:

- Name, power state, bus
- “Parent”, “children” of the object
- Tuneable parameters for the object

Many make sense only for devices

# Attributes

```
# ls /sys/devices/platform/i2c_omap.1/i2c-1
1-0048  delete_device  new_device
1-0049  device         power
1-004a  i2c-dev        subsystem
1-004b  name           uevent
```

# I2C-Bus Devices

Inter-Integrated Circuit:

- Multi-master, single-ended serial bus
- Attaches low-speed peripherals to a host controller
- Attaches peripherals to each other
- Ideal for embedded systems (and very popular there!)

# I2C-Bus Devices

Bus interface:

- The part that connects the device to the bus

Device address:

- Unique for each device on a bus

# I2C-Bus Devices

Linux is always a master device:

- Other devices are slaves to Linux
- Other devices can be masters to each other
- (This is mostly an implementation issue)

# I2C-Bus Devices

```
# ls -F /sys/devices/platform/i2c_omap.1/i2c-1
1-0048/  delete_device  new_device
1-0049/  device@        power/
1-004a/  i2c-dev/       subsystem@
1-004b/  name           uevent
```

# I2C-Bus Devices

`.../devices/`

- It’s a *device*

`.../platform/`

- It’s a *platform* device

`.../i2c_omap.1/`

- The kobject itself

# I2C-Bus Devices

`.../i2c-dev/`

- It’s an *i2c host bus adapter* device
- (A virtual device with its own attributes)

```
# cat i2c-dev/i2c-1/name
OMAP I2C adapter
# cat i2c-dev/i2c-1/dev
89:1
```

# I2C-Bus Devices

```
.../1-0048/
```

- An attached device with address `0x48`

```
# ls 1-0048
driver  modalias  name  power  subsystem  twl4030_usb  uevent

# cat 1-0048/name
twl4030
```

# I2C-Bus Devices

```
# ls 1-0048/twl4030_usb
driver                         subsystem  vbus
microamps_requested_usb3v1  modalias
microamps_requested_usb1v5  power
microamps_requested_usb1v8  uevent

# cat 1-0048/twl4030_usb/vbus
off
```

# I2C-Bus Devices

“Can I turn vbus on?”

- Nope!

```
# ls -l 1-0048/twl4030_usb/vbus
-r--r--r-- 1 root root 4096 May 28 02:29 vbus
```

# I2C-Bus Devices

Communicating with I2C slaves:

- Not a function of sysfs
- Sysfs isn't an *interface*
- Interfaces use *device nodes*

Device nodes:

- `open(), close()`
- `read(), write()`
- `mmap(), ioctl()`

# I2C-Bus Devices

To communicate with a slave:

- Call `open()` on the adapter's device node
- Use `ioctl()` to specify chip address
- Use `ioctl()` to read, write the chip

`#include <i2c-dev.h>`

- For `i2c_smbus_read_byte()`, etc.
- See `lm-sensors` project, `i2c-tools` source code

# I2C-Bus Devices

```c
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

#include "i2c-dev.h"

int main (void)
{
  int file;
```

# I2C-Bus Devices

```c
int adapter_nr = 0;
char filename[20];

snprintf(filename, sizeof(filename),
         "/dev/i2c-%d", adapter_nr);
file = open(filename, O_RDWR);
if (file < 0) {
  perror("Could not open device");
  exit(1);
}
```

# I2C-Bus Devices

```

if (ioctl(file, I2C_SLAVE, addr) < 0) {
  perror("Could not set I2C_SLAVE");
  exit(2);
}

__s32 v = 0xdeadbeef;
```

# I2C-Bus Devices

```c
v = i2c_smbus_read_byte(file);
if (v < 0) {
  perror("i2c_smbus_read_word failed (2)");
  exit(3);
```

# GPIO Devices

Dual personalities:

- GPIO “chip”
- GPIO “pin”

The sysfs layout accommodates both

# GPIO Devices

```
# ls /sys/class/gpio
export       gpiochip160@  gpiochip64@
gpiochip0@   gpiochip192@  gpiochip96@
gpiochip128@ gpiochip32@   unexport

# ls -F .../gpiochip192
base       label   power/       uevent
device@    ngpio   subsystem@
```

# GPIO Devices

```
# cat .../base
192

# cat .../ngpio
20
```

# GPIO Devices

“Exporting” a GPIO pin:

- Each pin has a unique enumerator
- Creates an attribute directory
- Attributes to set pin direction, state

```
# echo 160 > .../export

# ls gpio160/
direction  power/       uevent
edge       subsystem@   value
```

# GPIO Devices

High vs. low:

- Write “1” or “0” to `value`
- (Ignored if pin is an input)

Input vs. output:

- The `direction` attribute
- Semantics address “initial value problem”

# GPIO Devices

```
# echo input > .../direction
# echo output > .../direction

# echo 1 > .../value
# echo 0 > .../value

# echo high > .../direction
# echo low > .../direction
```

# LEDs

Why have a separate API?

- Because We Can (tm)
- The LED might not be a GPIO!

Example:

- LED "triggers"

# LEDs

Types of triggers:

- Heartbeat
- MMC, NAND, ethernet activity
- Timer
- None

The list varies depending on config

# LEDs

```
# ls /sys/class/leds
beagleboard::pmu_stat    beagleboard::usr1
beagleboard::usr0

# ls -F .../beagleboard::usr1
brightness  max_brightness  subsystem@  uevent
device@     power/          trigger
```

# LEDs

```
# ls -l .../device
lrwxrwxrwx ... device -> ../../../leds-gpio

# cat .../trigger
none nand-disk [mmc0] heartbeat
```

# LEDs

Changing triggers:

- Can switch only among the available options

```
# echo heartbeat > .../trigger
```

# Input Devices

How "input" gets into the kernel:

- A specialized `char` device

Used by:

- Keyboards and mice
- Tablets and touch screens
- Accelerometers, gyroscopes...

# Input Devices

```
# ls -l /sys/class/input
event0 -> .../gpio-keys/input/input0/event0
event1 -> .../twl4030_pwrbutton/input/input1/event1
input0 -> .../gpio-keys/input/input0
input1 -> .../twl4030_pwrbutton/input/input1
mice -> .../virtual/input/mice
```

# Detecting Plug Events

`udev(8)`

- Waits for a change in `/sys/devices/` directory
- Scans attributes, decides what to do next

# Creating Attributes

```
#include <device.h>

int device_create_file(struct device *device,
          const struct device_attribute *entry);
```

# Creating Attributes

```
ssize_t show_me(struct device *dev,
                struct device_attribute *attr,
                char *buf)
{
    return sprintf(buf, ‘‘\%d’’, me);
}

ssize_t store_me(struct device *dev,
                 struct device_attribute *attr,
                 const char *buf, size_t count)
{
    me = simple_strtoul(buf, NULL, 10);
    return count;
}
```

# Creating Attributes

```
static DEVICE_ATTR(me, S_IWUSR | S_IRUGO, \
                   show_me, store_me);
...

int probe(struct device *dev)
{
    ...
    device_create_file(dev, &dev_attr_me);
    ...
}
```

# The Sysfs Virtual Filesystem

## Exploring the Linux Device Model

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer