Device Attributes

Introducing the Linux Device Model API

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer

The Linux Kernel’s Device Model

An abstraction for managing devices:

- What they are, where they are
- How they’re related to each other
- What services they offer to users
- How devices respond to state changes

The Linux Kernel's Device Model

Interfaces:

- No device nodes! Wait, ... what?!
- Not part of the device model per se

Alternatives:

- Use `miscdevice` or `cdev` as always
- Use "attributes" (preferred)

Device Attributes

Similar to device nodes, but:

- Tightly coupled to device, model
- Limited to one page of data
- Only “show” and “store” methods
- No streaming capability

Device Attributes

Advantages:

- No creation, removal of device nodes needed
- Scales well to large numbers of interface points

Disadvantages:

- No streaming
- Limited to `PAGE_SIZE`
- No `mmap()`

Device Attributes

Conventions:

- Strong association with a device
- Replacement for `ioctl()`
- Usually refer to tangible information
- Numbers are coded in base-10
- Single datum per attribute

Can encode information, and events

“Show” Method

Roughly a `read()`:

- Printable ASCII is preferred encoding

```
ssize_t foo_show_bar(struct device *dev,
                     struct device_attribute *attr,
                     char *buf)
{
    struct foo *foo = dev_get_drvdata(dev);
    return sprintf(buf, ‘‘%d\n", foo->bar);
}
```

“Store” Method

Roughly a `write()`:

- Printable ASCII is preferred encoding

“Store” Method

```
ssize_t foo_store_bar(struct device *dev,
                      struct device_attribute *attr,
                      const char *buf, size_t len)
{
    struct foo *foo = dev_get_drvdata(dev);
    int ret, bar;

    ret = strict_strtoul(buf, 16, &bar);
    if (ret < 0)
      return ret;

    foo->bar = bar;
    return 0;
}
```

strict_strtoul()

Strict parsing of incoming data:

- Typical `printf()`-style formats supported
- Several bases supported
- See `include/linux/kernel.h` for more options

```
int strict_strtoul(const char *s,
    unsigned int base, unsigned long *res);
```

DEVICE_ATTR

Helper macro:

- Defines a `device_attribute` structure
- See `linux/device.h` for variations

```
#define DEVICE_ATTR(_name, _mode, _show, _store) \
    struct device_attribute dev_attr_##_name = \
    __ATTR(_name, _mode, _show, _store)
```

DEVICE_ATTR

```
/* creates dev_attr_bar */
static DEVICE_ATTR(bar, S_IRUGO | S_IWUSR,
    foo_show_bar, foo_store_bar);
```

Registering an Attribute Interface

```
int foo_probe(struct platform_device *pdev)
{
    ...
    ret = device_create_file(&pdev->dev,
                             &dev_attr_bar.attr);
    ...
}

int foo_remove(struct platform_device *pdev)
{
    ...
    device_remove_file(&pdev->dev, &dev_attr_bar.attr);
    ...
}
```

Registering Multiple Attribute Interfaces

```
static struct attribute *foo_attrs[] = {
    &dev_attr_bar.attr,
    &dev_attr_baz.attr,
    &dev_attr_qux.attr,
    NULL,
};
static struct attribute_group foo_attr_group = {
    .attrs = foo_attrs,
};
```

Registering Multiple Attribute Interfaces

```
int foo_probe(struct platform_device *pdev)
{
    ...
    ret = sysfs_create_group(&pdev->dev.kobj,
                             &foo_attr_group);
    ...
}

int foo_remove(struct platform_device *pdev)
{
    ...
    sysfs_remove_group(&pdev->dev.kobj, &foo_attr_group);
    ...
}
```

Registering Multiple Attribute Interfaces

```
$ ls -CF /sys/bus/platform/devices/foo.0
...        bar        baz        qux        ...
$ cat /sys/bus/platform/devices/foo.0/qux
10
```

Device Attributes

Introducing the Linux Device Model API

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer