Linux Devices and Drivers

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 components:

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

Merely a set of data structures!

The Linux Kernel's Device Model

Motivations:

- Power management
- Orderly system startup and shutdown
- Communications with user space
- System configurability
- Smarter user applications

On Modeling Hardware

Never tell Linux what to do!

- Only tell Linux what you *need*
- Make Linux itself figure out how to meet that need

Secret to success:

- *Model* your platform properly to Linux
- (Hence the term “device model API”)

On Modeling Hardware

```
/* turn on LVSW14 */
struct regulator lvsw14
    = regulator_get(NULL, "LVSW14");    /* no! */
regulator_enable(lvsw14);               /* no! */

/* turn on our VDD pin */
struct regulator vdd
    = regulator_get(dev, "VDD");        /* yes! */
regulator_enable(vdd);                  /* yes! */
```

On Modeling Hardware

The Device Model API is big!

- Lots of code
- LOTS of data structures

Overwhelming, at first:

- Solves a complex problem
- Somewhat different mindset is required

On Modeling Hardware

But:

- You don't need to understand how it works!
- You just need to know how to use it well!

Essential elements:

- Proper terminology!
- Correct schemas!

“Split” Devices and Drivers

A *split* implementation model:

- “Driver” is separated from “device”
- Analogous to C++ class “methods” and “data”
- Promotes reuse, portability
- See `struct device_driver` and `struct device`

`struct device`

What is a “device”?

- Physical hardware component e.g. chip
- Sub-functions of a complex chip
- Instance of an abstract component e.g. “LED”, “GPIO”

See `struct device`:

- Represents instances of “devices”
- (You don’t often use this structure directly)

struct device

```
struct device {
   ...
   const char              *init_name;
   struct device           *parent;
   struct kobject           kobj;
   struct bus_type         *bus;
   struct device_driver    *driver;
   void                    *platform_data;
   ...
};
```

`struct device`

"Where is my `open()`, etc.?"

- Those aren't devices, they are *interfaces*
- Linux devices DON'T have interfaces!
- Linux devices are inanimate data objects

"No interfaces?!"

- Relax, they are trivially implemented :)

`struct device_driver`

What is a “device driver”?

- Depends on whom you ask!
- (Users are almost never aware of true Linux device drivers)

See `struct device_driver`:

- Animates instances of `struct device` objects
- (You don’t often use this structure directly)

struct device_driver

```
struct device_driver {
  int   (*probe   ) (struct device *dev);
  int   (*remove  ) (struct device *dev);
  void  (*shutdown) (struct device *dev);
  int   (*suspend ) (struct device *dev,
                     pm_message_t state);
  int   (*resume  ) (struct device *dev);
  ...
```

struct device_driver

```
    ...
    const char                      *name;
    const struct dev_pm_ops         *pm;
    struct module                   *owner;
    struct driver_private           *p;
    const struct attribute_group **groups;
    ...
};
```

`struct device_driver`

"Where is my `open()`, etc.?"

- Those aren't device drivers, they are *interfaces*
- Linux device drivers don't implement interfaces per se

"No interfaces?!"

- Relax, they are trivially implemented :)
- (But you probably want a "device attribute" instead)

Device “Probing” and Removal

The `.probe()` method:

- Invoked when a device, driver `.name` match occurs

```
...
ret = device_add(struct device *dev);

...
ret = driver_register(struct device_driver *driver);
```

“When do I register?”

Device drivers:

- Module initialization, usually
- During `do_initcalls()` otherwise

“When do I register?”

```
static struct device_driver foo = {
  ...
  .name = "foo",
  ...
};

static int __init foo_init(void)
{
  ...
  return driver_register(&foo);
}
module_init(foo_init);
```

“When do I register?”

Devices:

- Board startup, usually
- Can be done anytime, really

```
p = platform_device_alloc("foo", -1);
ret = platform_device_add(p);
```

Exercise

Linux Devices and Drivers

Introducing the Linux Device Model API

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer