Linux Driver Model

"web woven by a spider on drugs"

Greg Kroah-Hartman
gregkh@linuxfoundation.org

github.com/gregkh/presentation-driver-model

You don’t need to know this.

struct kref

- Reference counting
- No locks
- Release function required

struct kobjects

- Base object type
- sysfs representation
- Data structure glue
- Hotplug event handling

struct attribute

- sysfs files for kobjects
- 1 text value per file
- Binary files possible
- Never manage indivually

struct kset

- Groups kobjects together

kobj_type

- release()
- sysfs functions for kobject
- Namespace handling

struct device

- Universal structure
- Belongs to a bus or “class”

struct device_type

- Same bus, different type

struct device_driver

- controls a device
- probe/remove
- shutdown/suspend/resume
- Default attributes

struct bus_type

- binds devices and drivers
- matching
- uevents
- shutdown

bus responsibilities

- register bus
- create devices
- register drivers
- suspend/resume

Create a device

- set the bus type
- set parent pointer
- set attribute groups
- device_initialize()
- ..other stuff..
- device_add()

Register a driver

- set the bus type
- set up probe/release
- set module ownership
- driver_register()

struct class

- user visable devices
- yes, it’s a bus...
- suspend/resume
- release
- default attributes

class responsibilities

- class_create/class_register
- reserve major/minor
- use in device_create
- device_destroy
- free major/minor
- suspend/resume if wanted

shutdown

```
if device→class
   if class→shutdown
      device→class→shutdown(device)

if device→bus
   if bus→shutdown
      device→bus→shutdown(device)
      shutdown must call
         driver→shutdown(device)
```

Driver writer hints

- attribute groups only
- never call `sysfs_*()`

Class writer hints

- attribute groups only
- never call `sysfs_*()`

Bus writer hints

- My sincere appologies

“raw” sysfs/kobjects?

- Do not do it
- Really, no.
- Use a class or a bus
- Please no.
- Read the documentation
- Read it again
- Expect tough review cycle

github.com/gregkh/presentation-driver-model

Linux Driver Model

"web woven by a spider on drugs"

Greg Kroah-Hartman
gregkh@linuxfoundation.org

github.com/gregkh/presentation-driver-model

Subtitle is LWN quote

In 2.4 all subsystems were isolated and did things their own way. After adding hotplug support to the second subsystem, I figured this needed to change

Pat Mochel wanted to get suspend/resume to work, I wanted persistant device naming. By the time 2.6 came out, we had naming solved, still working on suspend/resume...

You don’t need to know this.

No driver write should have to mess with the driver core at all.

Well, minor things, see my last slide.

It’s a complex mess, is a whole chapter in LDD3, and there are parts of the kernel code that I keep forgetting and having to relearn at times.

Messy stuff.

struct kref

- Reference counting
- No locks
- Release function required

Don’t ever do your own reference counting

Kref is “proven” correct

Use it

You need a lock outside of the object, it must be there

If no release function, why are you doing reference counting?

struct kobjects

- Base object type
- sysfs representation
- Data structure glue
- Hotplug event handling

Handles all of the basic housekeeping for kernel objects

Handles all of the links to other kobjects, the heiarchy and other good stuff. Solid functions that are known good.

Created out of the development process by Al Viro during the driver core development and was originally used in char devices

NEVER touch the kobject in a char device structure, it doesn’t do what you think it does.

struct attribute

- sysfs files for kobjects
- 1 text value per file
- Binary files possible
- Never manage indivually

You will be yelled at if you don’t follow these rules.

Bad examples are a histogram graph plot by a cpufreq driver

Hopefully all fixed.

Always document them in Documentation/ABI

struct kset

- Groups kobjects together

A “set” of kobjects that belong to the same type of “subsystem”. Don’t have to be the same type of object.

How you group a kobject together

kobj_type

- release()
- sysfs functions for kobject
- Namespace handling

The functions to call for your kobject

A kobject always has a kset and a kobj_type
Both are needed.

Namespace stuff is for only networking
kobjects, messy stuff.

struct device

- Universal structure
- Belongs to a bus or “class”

Basic features of all types of devices in the kernel

Name, platform data, driver data, dma information, MSI information, CMA, firmware info, iommu, and so on

Always belongs to something, never have a “blank” device.

NEVER ABUSE PLATFORM DEVICES!

struct device_type

- Same bus, different type

Devices that are of the same bus, but do different things.

USB example – device, interface, endpoint, port

Everything is handled by the same bus, but sometimes you need to do minor different things based on the type.

struct device_driver

- controls a device
- probe/remove
- shutdown/suspend/resume
- Default attributes

Controls a specific type of device

What you are used to thinking about, but you don’t ever access this structure directly.

Busses wrap it with their “type” of driver.

struct bus_type

- binds devices and drivers
- matching
- uevents
- shutdown

How you group devices and drivers together.

Handles the matching of a device to a driver

Handles the uevents for hotplug

Can handle default shutdown functions for a bus

bus responsibilities

- register bus
- create devices
- register drivers
- suspend/resume

Busses do a lot of different things

It’s a lot of work to write a bus, sorry about this.

A small one can be done in 300 lines. Real ones end up being much larger.

Create a device

- set the bus type
- set parent pointer
- set attribute groups
- device_initialize()
- ..other stuff..
- device_add()

A bus has to do all of this for when it creates a new device.

If you don’t want to do device init/add, you can just do device_register

You want to do other stuff sometimes like figuring out what sysfs files to add for the device before you announce it to the rest of the world.

So most “real” busses do the two step process.

Register a driver

- set the bus type
- set up probe/release
- set module ownership
- driver_register()

A bus also manages drivers for that bus

Has to handle registering and unregistering the drivers, setting up the needed pointers and other fun stuff.

struct class

- user visable devices
- yes, it’s a bus...
- suspend/resume
- release
- default attributes

“struct class” - my crowning achievement

A group of struct device.

These are the things that users are used to seeing.

Input, gpio, pwm, misc, tty, block, and so on

Almost always has a device node to interact with userspace.

Is not hardware-dependant

Like a bus, but not really, there are no drivers for class devices.

Can handle suspend/resume and release for a class device

class responsibilities

- class_create/class_register
- reserve major/minor
- use in device_create
- device_destroy
- free major/minor
- suspend/resume if wanted

Much simpler list of things that a class has to do to mange devices.

Only about 30 lines of code.

Much easier than a bus.

shutdown

```
if device→class
  if class→shutdown
    device→class→shutdown(device)

if device→bus
  if bus→shutdown
    device→bus→shutdown(device)
    shutdown must call
      driver→shutdown(device)
```

Busses and classes and drivers all interact.

Example, shutdown.

When the system wants to shutdown, it walks the list of devices and does the following.

If a device has a class, then it calls the shutdown for it

If a device has a bus, then the bus is called and told to call shutdown, which then finds the bus driver for that device and then calls shutdown.

Messy, but it works...

Driver writer hints

- attribute groups only
- never call `sysfs_*()`

Never create sysfs files individually.

Or at all, attribute groups should be handled by your class or bus. If not, something is almost always wrong and needs to be fixed. Talk to me about this.

Never call any sysfs functions in a driver, or really, any driver core functions.

Only exception, sysfs_notify() for when an attribute value changes that someone was calling select/poll on. Pretty rare.

Class writer hints

- attribute groups only
- never call `sysfs_*()`

Never create sysfs files individually.

Use class_create()

Bus writer hints

- My sincere appologies

It’s not easy, I know.

More can be done to make it easier. Help is always appreciated, been on my TODO list for over 5 years.

We add about 1-2 new busses every kernel release so there’s almost always some kernel devloper mad at me.

“raw” sysfs/kobjects?

- Do not do it
- Really, no.
- Use a class or a bus
- Please no.
- Read the documentation
- Read it again
- Expect tough review cycle

Ok, filesystems have to do it.

Firmware platforms had to do this.

It’s hard, rough, and easy to get wrong.

You thought writing a bus was difficult…

Loads of boilerplate code.

Obligatory Penguin Picture