

Coccinelle: A Language-Based Approach to Managing the Collateral Evolution of Linux Device Drivers

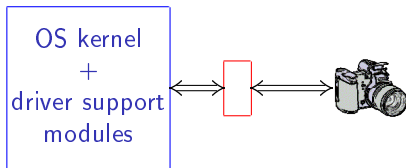
Julia Lawall

DIKU, University of Copenhagen

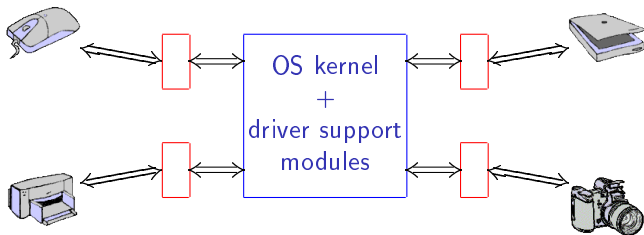
Gilles Muller, Yoann Padioleau

École des Mines de Nantes-INRIA, LINA

The device driver problem



The device driver problem (a more realistic view)

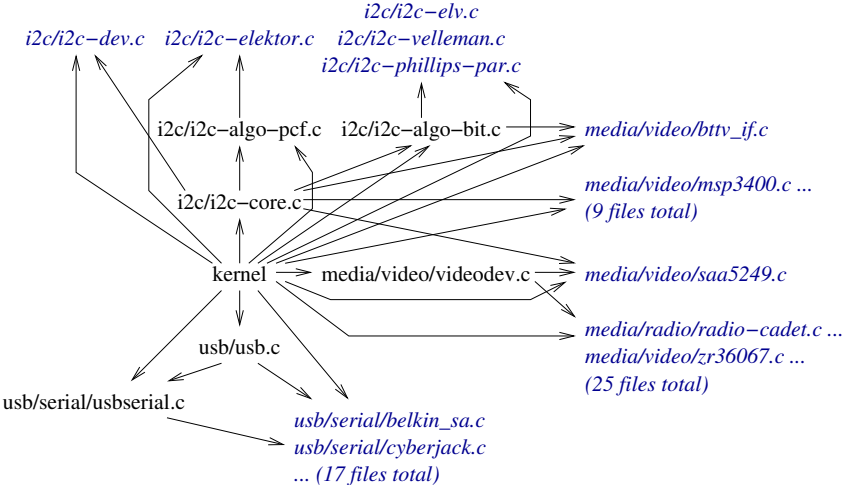


- ▶ OS evolves to improve performance, meet new hardware requirements, etc.
- ▶ Evolutions in the OS entail **collateral evolutions** in device drivers.

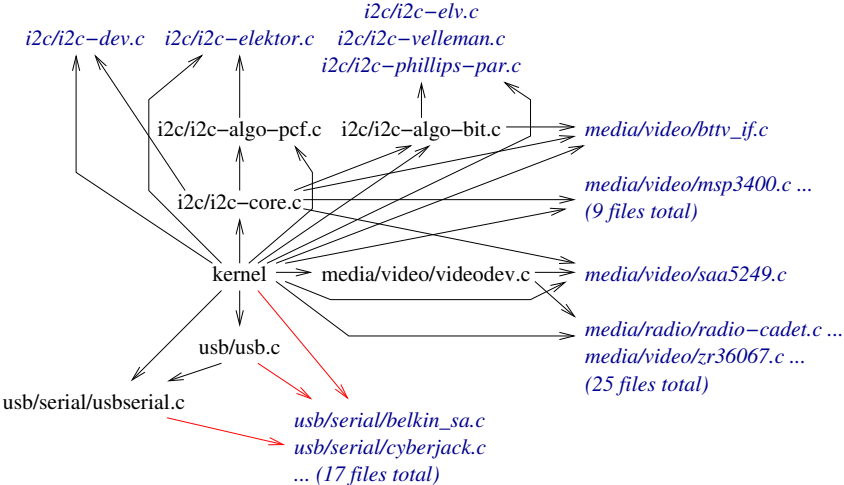
Collateral evolution and device drivers

- ▶ Many device drivers.
 - ▶ More than 70% of a modern OS [Engler - SOSP01]
- ▶ Complex dependencies between drivers and generic modules.
 - ▶ Families, subfamilies, cross-family relationships
- ▶ ...

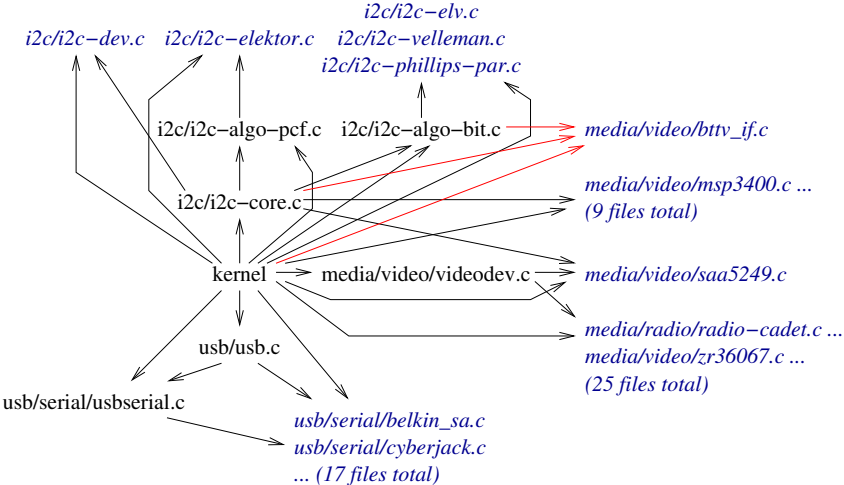
Linux driver dependencies



Linux driver dependencies



Linux driver dependencies



Collateral evolution and device drivers

- ▶ Many device drivers.
 - ▶ More than 70% of a modern OS [Engler - SOSP01]
- ▶ Complex dependencies between drivers and generic modules.
 - ▶ Families, subfamilies, cross-family relationships
- ▶ Varying expertise of driver maintainers.
 - ▶ Developer performing the evolution, driver developer, user.
- ▶ Orphaned drivers.
- ▶ Drivers that evolve outside the kernel source tree.

Collateral evolution is **slow** and **error-prone**.

A tempting target for automatization

Refactoring

- ▶ A range of common correctness preserving transformations
- ▶ Requires the whole program
- ▶ Not programmable

Aspect-Oriented Programming

- ▶ Extension with new functionalities (quota, etc. [Coady, et al.])
- ▶ Programmable, but traditionally non-invasive

Transformation languages

- ▶ Stratego provides transformation-specific abstractions, but ultimately is low-level

Our proposal: Coccinelle

- ▶ Transformation language & interactive transformation tool
- ▶ Targeted to the requirements of device drivers

Overview

- ▶ OS interfaces and their possible evolutions
- ▶ Collateral evolutions in device drivers
- ▶ Requirements on Coccinelle
- ▶ Using Coccinelle
- ▶ Current status, future work, and conclusions

OS-driver interfaces (implicit)

Generic functions exported by the OS.

Callbacks provided by each device driver.

Data structures for communicating between the OS and the device driver.

The protocol describing what can be used when.

Evolutions in the OS-driver interfaces

Generic functions exported by the OS.

- ▶ Function name, arguments, and return value.

Callbacks provided by each device driver.

- ▶ Function parameters and return value.

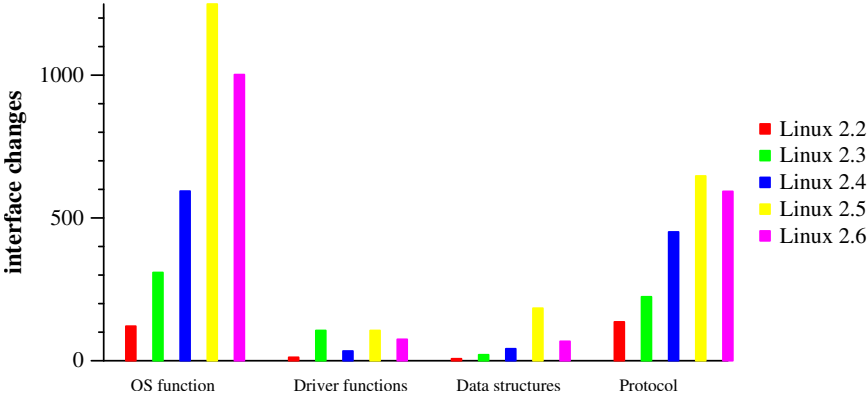
Data structures for communicating between the OS and the device driver.

- ▶ Structure layout, “public” vs. “private” fields.

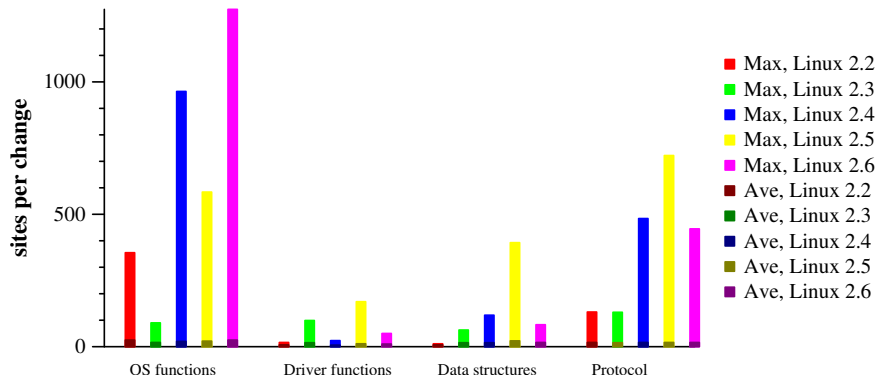
The protocol describing what can be used when.

- ▶ Function calls added, removed, ordering changed.
- ▶ Definitions moved from the OS to the driver, or vice versa.

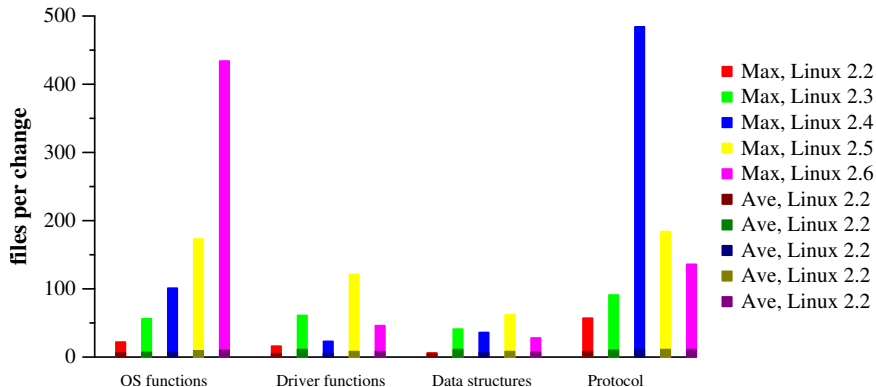
Interface changes



Sites per change



Files per change



Collateral evolutions in drivers (examples)

Generic functions exported by the OS.

- ▶ Driver must create new arguments, use new return values.

Callbacks provided by each device driver.

- ▶ Driver must reconstruct missing parameters.

Data structures for communicating between the OS and the device driver.

- ▶ Calls to getter functions may be factorized.

The protocol describing what can be used when.

- ▶ Analysis required to determine how the new protocol maps to driver code.

Examples

`check_region` elimination

- ▶ Change arguments and return type.
- ▶ Change protocol.

An extra argument for `usb_submit_urb`

- ▶ Construct new argument.

Check_region elimination

Original driver initialization pattern:

```
if (check_region(region,size))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;  
request_region(region,size, "name");
```

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (check_region(region,size))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;  
request_region(region,size, "name");
```

- ▶ `check_region` → `request_region`

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (request_region(region,size,"name"))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;
```

- ▶ `check_region` → `request_region`

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (request_region(region,size,"name"))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;
```

- ▶ `check_region` → `request_region`
- ▶ Adjust the return value.

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (!request_region(region, size, "name"))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;
```

- ▶ `check_region` → `request_region`
- ▶ Adjust the return value.

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (!request_region(region, size, "name"))  
    return FAIL;  
if (the_device_is_not_my_device())  
    return FAIL;
```

- ▶ `check_region` → `request_region`
- ▶ Adjust the return value.
- ▶ Insert calls to `release_region`.

Check_region elimination

Parallel driver initialization pattern (as of Linux 2.4.2):

```
if (!request_region(region,size,"name"))
    return FAIL;
if (the_device_is_not_my_device())
    { release_region(region,size); return FAIL; }
```

- ▶ `check_region` → `request_region`
- ▶ Adjust the return value.
- ▶ Insert calls to `release_region`. **Difficult!**

Help provided to developers

Subject: Re: Linux-2.6.13 : __check_region is deprecated

Newsgroups: gmane.linux.kernel

Date: 2005-08-29 23:21:30 GMT

On Tue, 30 Aug 2005, Stephane Wirtel wrote:

> Hi,

>

> By compiling my kernel, I can see that the __check_region function (in
> kernel/resource.c) is deprecated.

>...

> Is there a function to replace this deprecated function ?

Just restructure the code to use request_region().

Example (i2c-piix4.c, Linux 2.5.65)

```
static int piix4_setup(...) {
    int error_return = 0;
    ...
    if(ibm_dmi_probe()) {
        dev_err(...); error_return = -EPERM; goto END;
    }
    ...
    if (check_region(piix4_smba, 8)) {
        dev_err(...); error_return = -ENODEV; goto END;
    }
    ...
    if (force_addr) {...}
    else if ((temp & 1) == 0) {
        if (force) {...}
        else { dev_err(...); error_return = -ENODEV; goto END; }
    }
    request_region(piix4_smba, 8, "piix4-smbus");
    ...
END: return error_return;
}
```

Example (i2c-piix4.c, Linux 2.5.66)

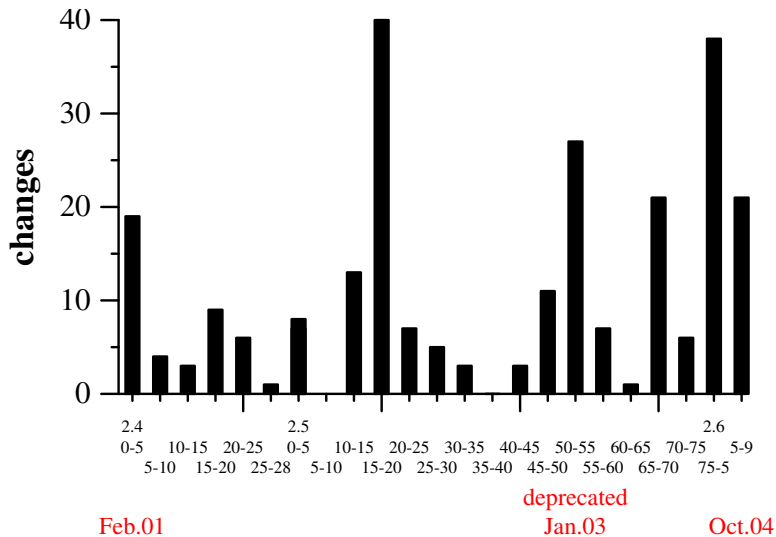
```
static int piix4_setup(...) {
    int error_return = 0;
    ...
    if(ibm_dmi_probe()) {
        dev_err(...); error_return = -EPERM; goto END;
    }
    ...
    if (!request_region(piix4_smba, 8, "piix4-smbus")) {
        dev_err(...); error_return = -ENODEV; goto END;
    }
    ...
    if (force_addr) {...}
    else if ((temp & 1) == 0) {
        if (force) {...}
        else { dev_err(...); error_return = -ENODEV; goto END; }
    }
    ...
END: return error_return;
}
```

Example (i2c-piix4.c, Linux 2.5.66)

```
static int piix4_setup(...) {
    int error_return = 0;
    ...
    if(ibm_dmi_probe()) {
        dev_err(...); error_return = -EPERM; goto END;
    }
    ...
    if (!request_region(piix4_smba, 8, "piix4-smbus")) {
        dev_err(...); error_return = -ENODEV; goto END;
    }
    ...
    if (force_addr) {...}
    else if ((temp & 1) == 0) {
        if (force) {...}
        else { dev_err(...); error_return = -ENODEV; goto END; }
    }
    ...
END: return error_return;
}
```

Error fixed in Linux 2.6.2

The slow pace of evolution



Requirements on Coccinelle

Connect `check_region` call to `request_region` call

- ▶ Flow analysis.
- ▶ Possibly interprocedural.

Identify error paths

- ▶ Paths returning 0?
- ▶ Paths returning `-ENODEV`, etc?
- ▶ Paths never reaching `request_region`?
 - ▶ Requires interprocedural analysis with propagation of return values.

An extra argument for `usb_submit_urb`

`usb_submit_urb`:

- ▶ USB message passing (`urb` = USB request block)
- ▶ Uses `kmalloc`

Evolution:

- ▶ Starting in Linux 2.5.4, `usb_submit_urb(urb)` becomes
 - ▶ `usb_submit_urb(urb, GFP_KERNEL)`
 - ▶ `usb_submit_urb(urb, GFP_ATOMIC)`
 - ▶ `usb_submit_urb(urb, GFP_NOIO)`

How to choose the new argument?

Choosing GFP_ATOMIC

GFP_ATOMIC required:

- ▶ in a completion handler
- ▶ in an interrupt handler
- ▶ when locks are held
- ▶ when the running process may block
- ▶ in some network driver functions
- ▶ in SCSI driver `queuecommand` functions

Example (usb/class/audio.c, Linux 2.5.4 – Linux 2.6.11)

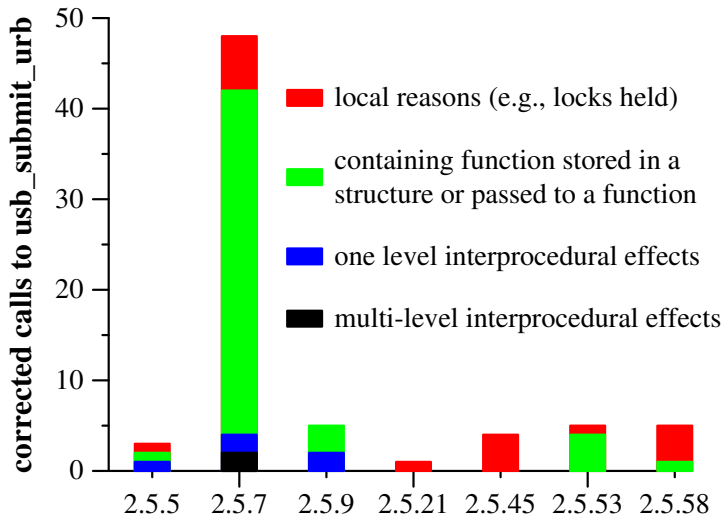
```
static int usbin_start(struct usb_audiodev *as) {
    ... // declarations
    spin_lock_irqsave(&as->lock, flags);
    if (!(u->flags & FLG_CONNECTED)) {
        spin_unlock_irqrestore(&as->lock, flags);
        return -EIO;
    }
    if (!(u->flags & FLG_RUNNING)) {
        spin_unlock_irqrestore(&as->lock, flags);
        ... // 28 lines of conditionals and straightline code
        spin_lock_irqsave(&as->lock, flags);
    }
    ...
    u->flags |= FLG_RUNNING;
    if (!(u->flags & FLG_URBORUNNING)) {
        ... // various assignments
        if (!usbin_prepare_desc(u, urb) && !usb_submit_urb(urb, GFP_KERNEL))
            u->flags |= FLG_URBORUNNING;
        else u->flags &= FLG_RUNNING;
    }
    ... // 3 copies of the preceeding code
    spin_unlock_irqrestore(&as->lock, flags);
    return 0;
}
```

Example (usb/class/audio.c, Linux 2.5.4 – Linux 2.6.11)

```
static int usbin_start(struct usb_audiodrv *as) {
    ... // declarations
    spin_lock_irqsave(&as->lock, flags);
    if (!(u->flags & FLG_CONNECTED)) {
        spin_unlock_irqrestore(&as->lock, flags);
        return -EIO;
    }
    if (!(u->flags & FLG_RUNNING)) {
        spin_unlock_irqrestore(&as->lock, flags);
        ... // 28 lines of conditionals and straightline code
        spin_lock_irqsave(&as->lock, flags);
    }
    ...
    u->flags |= FLG_RUNNING;
    if (!(u->flags & FLG_URBORUNNING)) {
        ... // various assignments
        if (!usbin_prepare_desc(u, urb) && !usb_submit_urb(urb, GFP_KERNEL))
            u->flags |= FLG_URBORUNNING;
        else u->flags &= FLG_RUNNING;
    }
    ... // 3 copies of the preceding code
    spin_unlock_irqrestore(&as->lock, flags);
    return 0;
}
```

The slow pace of correct evolution

71 errors among 158 call sites



Requirements on Coccinelle

Identify enclosing taking and releasing of lock

- ▶ Flow analysis.

Analyze the use of the enclosing function
(stored in a structure, called with locks held, etc.)

- ▶ Interprocedural analysis.
- ▶ Alias analysis.

Other requirements and non-requirements

Requirements

- ▶ A means of describing driver constructs in terms of their relationships.
 - ▶ Defined functions: how is the function exported.
 - ▶ Variables: how is the variable used elsewhere.
- ▶ Abstraction of related operations: error return, lock, etc.
- ▶ Other transformations: specialization, factorization of common subexpressions, etc.

Non-requirements

- ▶ Recursion.
- ▶ Exceptions.

Using Coccinelle

A developer who makes an evolution writes a rewrite rule describing the corresponding collateral evolution

- ▶ Rules describe transformations and detect incomplete matches of expected patterns.

Still, complete automation is unrealistic

- ▶ Some new code is hard to anticipate (e.g., error codes).
- ▶ Some rules may not apply in driver-specific conditions
 - ▶ Rare, because the driver must respect the new interface.
- ▶ Rules are limited by the developer's experience, imagination, and patience, and by the rule language's expressiveness.

Our proposal

Interactive rule application

- ▶ The developer tests and iteratively refines the rule on drivers in the kernel source tree.
- ▶ The rule is then published for use outside the kernel source tree.

Current status

- ▶ ~50 collateral evolutions studied in detail.
- ▶ Hundreds more probable collateral evolutions identified.
- ▶ Tools for evolution detection under development.
- ▶ Rule language under development.

Conclusion

Keeping drivers up to date with respect to evolutions in generic modules is a difficult task

- ▶ Many sites require collateral evolutions.
- ▶ Complex analyses required:
 - ▶ Control-flow analysis, alias analysis, etc.
- ▶ The requirements on the evolution may be insufficiently documented.
- ▶ Errors are introduced, and may persist.

Our proposal: Coccinelle

- ▶ A language for specifying collateral evolutions.
- ▶ A rewriting engine for interactively applying collateral evolutions.

Greg Kroah-Hartman has gotten [Linux] 2.6.13 off to a good start with a massive set of driver core patches. There are a fair number of API changes that come with this patch set, so the whole thing is worth a look. In-tree code has been fixed to use the new API, but, as always, maintainers of external code are on their own.
<http://lwn.net/Articles/140002/>, June 23, 2005.