Coccinelle: Killing Driver Bugs Before They Hatch

Julia Lawall
DIKU, University of Copenhagen

Gilles Muller, Richard Urunuela
École des Mines de Nantes-INRIA, LINA

OS evolution

Motivations:

- Improve performance.
- Meet new hardware requirements.
- Improve the software architecture.

Effects:

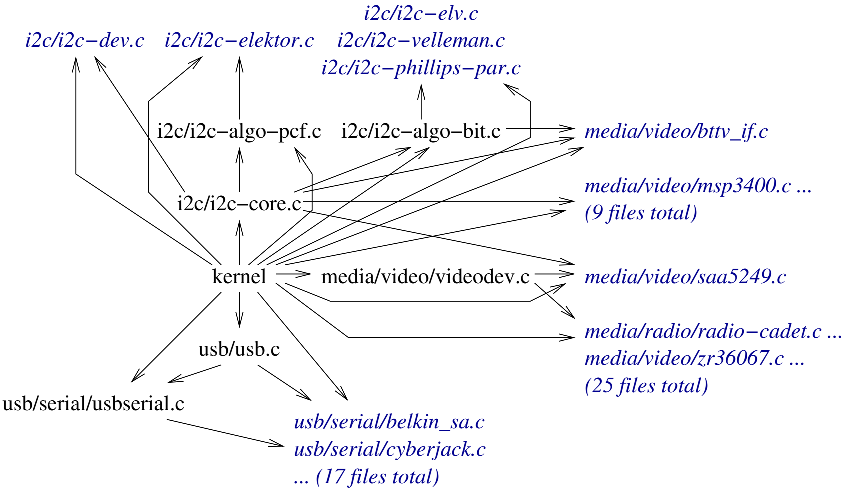
- Evolutions can affect the interface between modules.
  - Exported functions, data structures.
- Evolutions in generic modules can cause collateral evolutions in more specific modules.
  - Generic module: usb/serial/usbserial.c
  - More specific modules: usb/serial/{belkin_sa.c,cyberjack.c,...}

Collateral evolution and device drivers

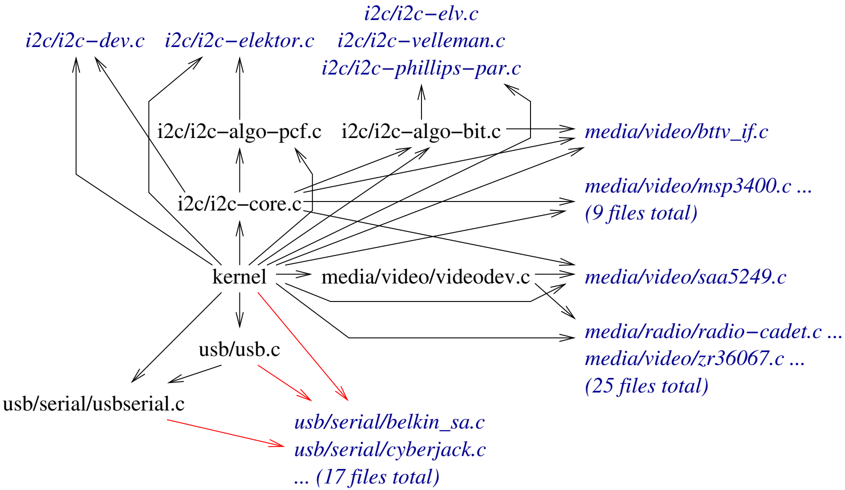
Problems for collateral evolution:

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

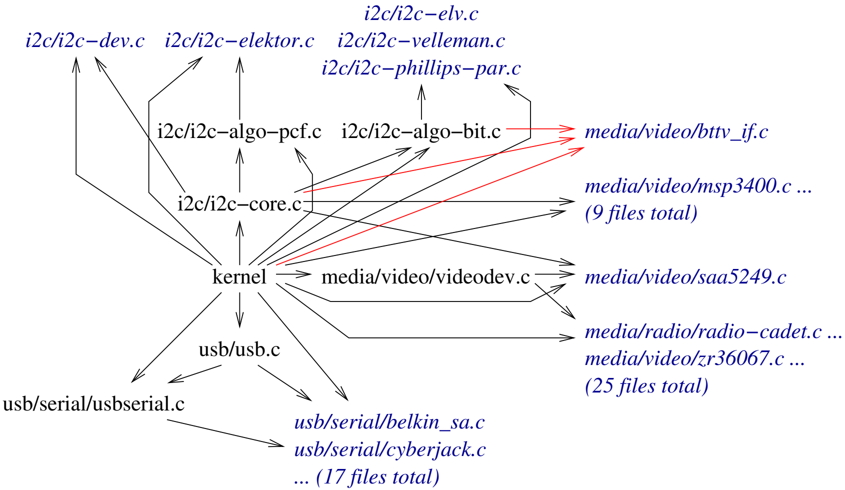
Linux driver dependencies



Linux driver dependencies



Linux driver dependencies



Collateral evolution and device drivers

Problems for collateral evolution:

- 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.

Our goal

Coccinelle:

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

This talk:

- A study of evolutions in generic modules.
- A study of collateral evolutions in drivers.
  - What kinds of transformations are required by collateral evolutions?
  - Why is collateral evolution difficult?
- An assessment of the requirements on Coccinelle.

Evolutions and collateral evolutions: examples

Change a function signature (new name, arguments, return type)

- Call site: create new arguments, use new return value.
- Function body: recreate dropped arguments, use new ones.

Add a required initialization, finalization

- Add initialization to `init` functions.
- Add finalization to `exit` functions and init failure code.

Reorganize a structure

- Identify and rewrite structure accesses.

Introduce new coding conventions (error handling, etc.)

- Potentially pervasive, tedious, and complex changes.

Examples

check_region elimination

- Change return type.
- Introduce finalization code.

An extra argument for usb_submit_urb

- Construct new argument.

video_usercopy introduction

- Change variable type.

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!

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;
}
```

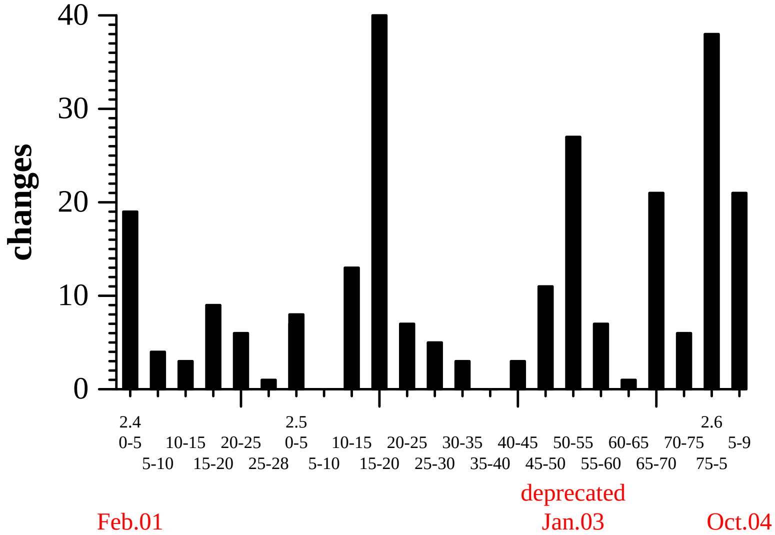
Example (i2c-piix4.c, Linux 2.5.66)

```
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)

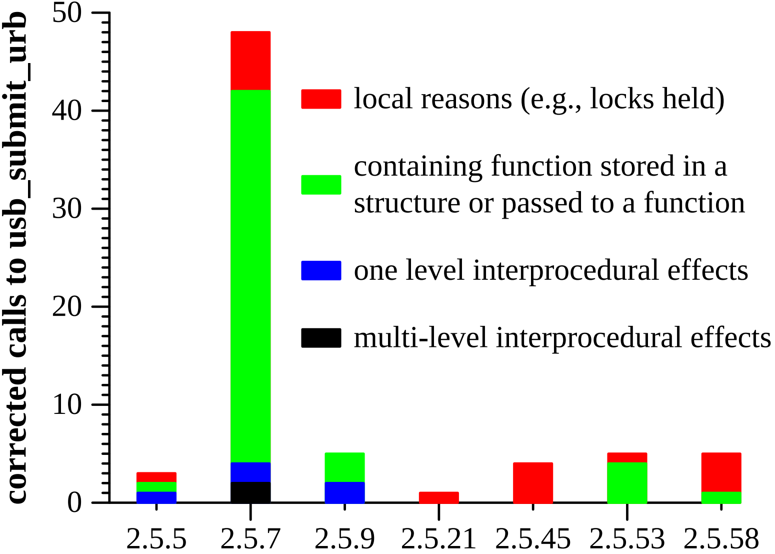
```
static int usbin_start(struct usb_audiodev *as) {
  ... // declarations
  spin_lock_irqsave(&as->lock, flags);
  if (!(u->flags & FLG_CONNECTED)) {
    spin_unlock_irqrestore(&as->lock, flags);
    return -EIO;
  }
  if (!(u->flags & FLG_RUNNING)) {
    spin_unlock_irqrestore(&as->lock, flags);
    ... // 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_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;
}
```

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.

Video_usercopy introduction

Standard IOCTL pattern (radio-typhoon.c, Linux 2.5.6):

```
static int typhoon_ioctl(struct video_device *dev,
                         unsigned int cmd, void *arg) {
  struct typhoon_device *typhoon = dev->priv;
  switch (cmd) {
  case VIDIOCGTUNER: {
    struct video_tuner v;
    if (copy_from_user(&v, arg, sizeof(v)) != 0) return -EFAULT;
    if (v.tuner) return -EINVAL;
    v.rangelow = 875 * 1600;
    ...
    if (copy_to_user(arg, &v, sizeof(v))) return -EFAULT;
    return 0;
  }
  case VIDIOCSTUNER: { ...}
  ...
}
```

Video_usercopy IOCTL pattern (Linux 2.5.8)

```
static int typhoon_ioctl(struct video_device *dev,
                         unsigned int cmd, void *arg) {

  struct typhoon_device *typhoon = dev->priv;
  ...
}
```

```
static int typhoon_ioctl(struct inode *inode, struct file *file,
                         unsigned int cmd, unsigned long arg) {
  return video_usercopy(inode, file, cmd, arg, typhoon_do_ioctl);
}
```

Video_usercopy IOCTL pattern (Linux 2.5.8)

```
static int typhoon_do_ioctl(struct inode *inode, struct file *file,
                           unsigned int cmd, void *arg) {
  struct video_device *dev = video_devdata(file);
  struct typhoon_device *typhoon = dev->priv;
  ...
}
```

```
static int typhoon_ioctl(struct inode *inode, struct file *file,
                        unsigned int cmd, unsigned long arg) {
  return video_usercopy(inode, file, cmd, arg, typhoon_do_ioctl);
}
```

Using the new arg value

- Standard IOCTL pattern: arg is a user pointer.
- Video_usercopy pattern: arg is a kernel pointer.

```
struct video_tuner v;
if (copy_from_user(&v, arg, sizeof(v)) != 0) return -EFAULT;
if (v.tuner) return -EINVAL;
v.rangelow = 875 * 1600;
...
if (copy_to_user(arg, &v, sizeof(v))) return -EFAULT;
return 0;
```

becomes:

```
struct video_tuner *v = arg;
if (v->tuner) return -EINVAL;
v->rangelow = 875 * 1600;
...
return 0;
```

typhoon_ioctl has ∼90 lines. 61% are affected.

Requirements on Coccinelle

Find the IOCTL function

- Analysis of global structures

Drop calls to copy functions

- Pattern matching, collect argument information.

Change local structure types

- Modification of local variable declarations and uses.

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

- $\sim$30 collateral evolutions studied in detail.
- $\sim$50 more probable collateral evolutions identified.
- 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.