Typical mistakes when submitting a new code to Linux kernel

Andy Shevchenko <andriy.shevchenko@intel.com>

June 23th, 2017.
Agenda

- Linux kernel is growing
- Managed Device Resources
- Unified Device Properties
- Special extensions of %p
- Perfect is the enemy of the good enough
- Recommendations how to prepare changes to Linux kernel
- Q&A
Linux is growing

Introduction or Linux is growing
Linux is growing: The most common issues

- Poor knowledge of the existing APIs
  - Internal APIs
  - APIs of the existing frameworks
  - Many small but not least helper functions

- Experience with only single architecture or platform
  - Device Tree is solely for ARM?
  - ACPI is solely for x86?
  - There are no more Big Endian CPUs in use?

- Power management
  - Little understanding how it works
  - Interrupts can be all threaded
Linux is growing: New Developer vs. new helper function

```c
> > +
> > + for (i = 0; i < ARRAY_SIZE(lp_supported); i++) {
> > +     if (strcmp(synth->name, lp_supported[i]) == 0)
> > +         break;
> > + }
> > +
> > + if (i >= ARRAY_SIZE(lp_supported)) {
> > +     match_string()
> > + }
Cool, didn't know about it
Managed Device Resources

Few words about Managed Device Resources API
Managed Device Resources: Motivation

- Error path in the ->probe() callback might be twisted up
  - Hard to catch a logic mistake in case of error
  - Possible leak of resources

- Make developers’ life easier
  - No need to reinvent a wheel
  - Concentrate on the logic of the driver
  - Bugs, if any, are getting fixed faster and in one place
  - Add code in the middle of ->probe() callback is simple
  - Easy integration into existing code
Managed Device Resources: API

Memory management

- Memory allocation
  - devm_kasprintf()
  - devm_kcalloc(), devm_kmalloc_array()
  - devm_kmalloc(), devm_kzalloc
  - devm_kmemdup(), devm_kstrdup()

- IO mapping
  - devm_ioport_map()
  - devm_ioremap()
  - devm_ioremap_resource()

- DMA
  - dmam_alloc_coherent(), dmam_alloc_noncoherent()
  - dmam_pool_create()

Other resources

- IRQ
  - devm_request_irq(), devm_request_threaded_irq()

- PCI
  - pcim_enable_device()
  - pcim_iomap(), pcim_iomap_regions()
  - pcim_iomap_table()

- GPIO and pin control:
  - devm_gpiod_get(), devm_pinctrl_get()

- Industrial IO (IIO) bus
  - devm_iio_device_alloc()
  - devm_iio_device_register()
  - devm_iio_trigger_alloc()
Managed Device Resources: Example (19 LOCs)

```c
ret = of_address_to_resource(np, 0, &res_xbar);
if (ret) {
    dev_err(dev, "Failed to get xbar resources");
    return ret;
}

if (!devm_request_mem_region(dev, res_xbar.start,
       resource_size(&res_xbar),
       res_xbar.name)) {
    dev_err(dev, "Failed to get xbar resources");
    return -ENODEV;
}

xbar_membase = devm_ioremap_nocache(dev, res_xbar.start,
       resource_size(&res_xbar));
if (!xbar_membase) {
    dev_err(dev, "Failed to remap xbar resources");
    return -ENODEV;
}
```
Managed Device Resources: Example (5 LOCs)

```c
+ res_xbar = platform_get_resource(pdev, IORESOURCE_MEM, 0);
+ xbar_membase = devm_ioremap_resource(dev, res_xbar);
+ if (IS_ERR(xbar_membase))
+     return PTR_ERR(xbar_membase);
```
Unified Device Properties

Few words about Unified Device Properties API
Unified Device Properties: Motivation

- Three common resource providers
  - Device Tree
  - ACPI (especially r5.1 and newer)
  - (Legacy) platform data or board files

- Unification
  - Resource provider agnostic API
  - Code deduplication

- Bye, bye, platform data
  - PWM, GPIO provide lookup tables
  - The built-in device properties API
Unified Device Properties: API

Firmware node (Frameworks)

- **Boolean**
  - fwnode_property_present()
  - fwnode_property_read_bool()

- **Integer types**
  - fwnode_property_read_uXX()
  - fwnode_property_read_uXX_array()

- **Strings**
  - fwnode_property_read_string()
  - fwnode_property_read_string_array()
  - fwnode_property_match_string()

Device node (Drivers)

- **Boolean**
  - device_property_present()
  - device_property_read_bool()

- **Integer types**
  - device_property_read_uXX()
  - device_property_read_uXX_array()

- **Strings**
  - device_property_read_string()
  - device_property_read_string_array()
  - device_property_match_string()
Unified Device Properties: Conversion example (clean up of NFC pn544 driver: +62 –210 LOCs)

e7f6ccaab127 Get rid of platform data
1 file changed, 6 insertions(+), 37 deletions(-)

e2c518c6c998 Convert to use GPIO descriptor
1 file changed, 33 insertions(+), 93 deletions(-)

182d4e860845 Convert to use devm_request_threaded_irq()
1 file changed, 5 insertions(+), 11 deletions(-)

95129b6f0806 Get rid of code duplication in ->probe()
1 file changed, 17 insertions(+), 67 deletions(-)

38d4d2bb7119 Switch to devm_acpi_dev_add_driver_gpios()
1 file changed, 1 insertion(+), 2 deletions(-)
Special extensions of %p

Few words about special extensions of %p
Special extensions of %p: The list of (v4.11)

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Description</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols/Function Pointers</td>
<td>%p[FfSsB]</td>
<td>UUID/GUID</td>
<td>%pU[LIBb]</td>
</tr>
<tr>
<td>Kernel Pointers</td>
<td>%pK</td>
<td>Directory entry names</td>
<td>%p[Dd][234]</td>
</tr>
<tr>
<td>struct resources</td>
<td>%p[Rr]</td>
<td>Block device names</td>
<td>%pg</td>
</tr>
<tr>
<td>Physical addresses types</td>
<td>%pa[dp]</td>
<td>struct va_format</td>
<td>%pV</td>
</tr>
<tr>
<td>Raw buffer as an escaped string</td>
<td>%[*0-9]*pE[achnops]</td>
<td>Content of struct clk</td>
<td>%pC[nr]</td>
</tr>
<tr>
<td>Raw buffer as a hex string</td>
<td>%[*0-9]*ph[CDN]</td>
<td>Bitmap and its derivatives such as cpumask and nodemask</td>
<td>%[*0-9]*pb</td>
</tr>
<tr>
<td>MAC/FDDI addresses</td>
<td>%p[Mm][FR]</td>
<td>Flags bitfields such as page flags, GFP flags</td>
<td>%pG</td>
</tr>
<tr>
<td>IP addresses</td>
<td>%p[li][46S][pfschnbl]</td>
<td>Network device features</td>
<td>%pNF</td>
</tr>
</tbody>
</table>
Special extensions of %p: Least used ones (v4.11)

- 6  Network device features
- 12 Content of struct clk
- 22 Flags bitfields such as page flags, GFP flags
- 28 Block device names
- 67 Raw buffer as an escaped string
Special extensions of %p: Most used ones (v4.11)

- 1789  MAC/FDDI addresses
- 757   IP addresses
- 614   Raw buffer as a hex string
- 381   Symbols/Function Pointers
- 364   Physical addresses types (phys_addr_t, dma_addr_t)
Special extensions of %p: Conversion example (clean up of wireless at76c50x driver -30 LOCs)

commit 44afb60f3927c6f732522a477eb77c9db83bd404
Author: Andy Shevchenko <andriy.shevchenko@linux.intel.com>
Date:   Wed Sep 5 11:52:32 2012 +0300

wireless: at76c50x: eliminate hex2str()

The hex2str() is substituted by '%*phD' specificator.

Signed-off-by: Andy Shevchenko <andriy.shevchenko@linux.intel.com>
Tested-by: Larry Finger <Larry.Finger@lwfinger.net>
Signed-off-by: John W. Linville <linville@tuxdriver.com>

---
drivers/net/wireless/at76c50x-usb.c | 54 ++++++++-------------------------------
1 file changed, 12 insertions(+), 42 deletions(-)
Perfect is the enemy of the good enough

Few words about special cases when simplification leads to regression
Perfect is the enemy of the good enough: Case study: devm_request_threaded_irq()

- **Rule of thumb**
  - Don’t use devm_request_irq() or devm_request_threaded_irq() if you are not clear with the details

- **Requires special attention to be paid**
  - Interrupt handlers can be invoked at any time until they are not explicitly unlinked

- **Tasklets are in a race with interrupt handlers**
  - There is a race condition when tasklet might be scheduled just enough ahead of the freeing IRQ
Perfect is the enemy of the good enough: Case study: devm_kzalloc() et al.

- **Scenario of a crash (character device)**
  - User loads a driver
  - Driver registers a device node
  - User opens the device node
  - User unbinds the driver
  - User closes the device node
  - KABOOM!

- **Attributes in sysfs**
  - Is there a problem?

- **What about debugfs?**
Bisectability!
Bisectability!
Bisectability!
Bisectability!
Recommendations how to prepare changes to Linux kernel

Few words about changes which are going to be submitted to upstream
Recommendations how to prepare changes to Linux kernel (basic rules):

- Follow the Coding Style and Submitting Patches guidelines
  - They include some common sense rules how to make code clean in the first place
- Use existing code
  - For a new driver it makes sense to look at the existing code from a known author
Recommendations how to prepare changes to Linux kernel (in addition to):

- Check the code against duplications
  - Many helper functions are already implemented as a part of Linux kernel internal API

- Take the material from the above slides into consideration when doing drivers

- Establish internal mailing list for review process if it’s not done yet
  - If you are working in a team it is always a good idea to have an internal mailing list dedicated to patch review

- Include a reviewer to the next round if you got some comments
  - Pay a respect to reviewers who volunteered to go through your code

- If in doubt, feel free to ask
  - Public mailing lists, forums, friends – do not hesitate to ask!
Thank you!

Questions and Answers