

2026-10-07

Constification of sysfs attribute structures

Presented to you by:

Thomas Weißschuh (Linutronix)

About me

- ▶ Kernel hobbyist since 2018
- ▶ Maintainer of
 - nolibc
 - multiple ChromeOS Embedded Controller drivers
 - multiple hwmon drivers
- ▶ Various Bug fixes
- ▶ Working for Linutronix on the kernel since 2024
 - vDSO
 - timekeeping/timers

Sysfs

- ▶ Filesystem representation of kernel objects.
- ▶ Each object has a directory.
- ▶ Object attributes show up as files.
- ▶ Allow introspection and interaction.

A USB device example

```
$ ls -l /sys/devices/pci0000:00/0000:00:08.1/0000:c1:00.3/usb1/
...
manufacturer
vendor
...
```

Sysfs attribute structures

```
/* include/linux/device.h */  
struct device_attribute {  
    struct attribute      attr; /* name and mode */  
    ssize_t               (*show)(struct device *dev,  
                                   struct device_attribute *attr,  
                                   char *buf);  
    ssize_t               (*store)(struct device *dev,  
                                   struct device_attribute *attr,  
                                   const char *buf, size_t count);  
};
```

Attribute group structures

```
/* include/linux/sysfs.h */
struct attribute_group {
    const char          *name;
    umode_t              (*is_visible)(struct kobject *,
                                       struct attribute *,
                                       int);

    /* NULL-terminated list */
    struct attribute     **attrs;
    /* ... binary attributes */
};
```

Sysfs attribute structures (2)

- ▶ Attributes are registered on whole “classes” of objects.
- ▶ Attributes are added from different places:
 - Driver core.
 - Subsystems and subsystems.
 - Drivers.
- ▶ Per-object modifications is done through `is_visible()` callbacks.

Problems from modifications by the developer

- ▶ Structures are global, shared between all instances.
- ▶ Structures may be used at any time due to events.
 - Modifications are racy.
- ▶ Driver core expects the attributes to be stable.

Problems from malicious modifications

- ▶ Callback modifications by attackers can divert control flow.
- ▶ Callbacks are easily reachable from userspace.
- ▶ What about Control flow integrity (CFI)?
 - There is only a limited amount of different callback types.
 - The callbacks cast with `container_of()` manually.
 - Undetectable by CFI.

Solution

- ▶ The structures do not actually have to be modifiable.
- ▶ Mark them all `const` and move them to read-only memory.
- ▶ Enforce this for new instances through the APIs.
- ▶ Drop legacy mutable APIs if possible.

Current state

- 0) Test balloon using `struct bin_attribute`. [DONE]
- 1) Sysfs core preparation. [DONE]
- 2) API infrastructure for the migration. [DONE]
- 3) Gradual Migration of the attribute structures. [IN PROGRESS]
 - Some users require special on-off changes.
 - Requires inspection of each usage site.
- 4) Removal of the non-const APIs. [TODO]
- 5) Removal of the `_const` API suffix. [TODO]
 - Hopefully in one big, mechanical commit.

Migration example, container_of()

```
/* before */
#define to_dev_attr(_attr) \
    container_of(_attr, struct device_attribute, attr)

/* after */
#define to_dev_attr(_attr) \
    container_of_const(_attr, struct device_attribute, attr)
```

Not specific to sysfs. Use container_of_const() everywhere!

Migration example, DEVICE_ATTR()

/ before */*

```
ssize_t foo_show(..., struct device_attribute *attr, ...);  
ssize_t foo_store(..., struct device_attribute *attr, ...);
```

```
static DEVICE_ATTR_RW(foo);
```

/ after */*

```
ssize_t foo_show(..., const struct device_attribute *attr, ...);  
ssize_t foo_store(..., const struct device_attribute *attr, ...);
```

```
static const DEVICE_ATTR_RW(foo);
```

Migration example, __ATTR()

/ before */*

```
ssize_t bar_show(..., struct device_attribute *, ...);
```

```
static const struct device_attribute bar = __ATTR_RO(bar);
```

/ after */*

```
ssize_t bar_show(..., const struct device_attribute *, ...);
```

```
static const struct device_attribute bar = __DEVICE_ATTR_RO(bar);
```

Migration example, custom attribute creation:

```
/* before */
```

```
ssize_t bar_show(..., struct device_attribute *, ...);
```

```
struct device_attribute *dev_attr = ...;
```

```
dev_attr.show = bar_show;
```

```
/* after */
```

```
ssize_t bar_show(..., const struct device_attribute *, ...);
```

```
struct device_attribute *dev_attr = ...;
```

```
dev_attr.show_const = bar_show;
```

Migration for other attribute types

- ▶ `struct kobj_attribute` works the same as `struct device_attribute`.
- ▶ Custom attribute types can be migrated in big-bang switchover.
- ▶ ... or use their own transition machinery.

Migration example, attribute list

```
/* before */
static struct attribute *attrs[] = [
    &dev_attr_foo.attr,
    &bar.attr,
    NULL
];

/* after */
static const struct attribute *const attrs[] = [
    &dev_attr_foo.attr,
    &bar.attr,
    NULL
];
```

Migration example, attribute group

/ before */*

```
umode_t baz_is_visible(..., struct attribute *attr, ...);
```

```
const struct attribute_group group = {  
    .attrs      = attrs,  
    .is_visible = baz_is_visible,  
};
```

/ after */*

```
umode_t baz_is_visible(..., const struct attribute *attr, ...);
```

```
const struct attribute_group group = {  
    .attrs_const      = attrs,  
    .is_visible_const = baz_is_visible,  
};
```

Complexity behind the scenes

- ▶ Issues with CFI storing different function pointers in a union.
- ▶ Keeping both pointers in each structure wastes memory.
- ▶ Compatibility with non-const and const callers.
- ▶ Handled automatically by the sysfs cores and helper macros.

Call to action

- ▶ All users of `sysfs.h`:
 - Mark your attribute instances as `const`.
- ▶ Subsystems with custom attributes taking `struct <custom_>attribute *`:
 - Mark the callback argument as `'const'`.
- ▶ Users of `device_attribute` or `kobj_attribute`:
 - Switch to new type-specific macros.
- ▶ New kernel contributors:
 - Help with the migration.
 - Talk to me for guidance and review.