# Brief `sysfs` Tutorial

Ben Marks

Spring 2016

# A Few Disclaimers

- I'm not a Linux kernel expert … if it segfaults, I'm sorry. (It works for me though!)
- There are lots of functions that do very similar things.
  - The code presented here worked for me.
  - There may be other ways to do the same thing.
  - There may be easier ways to to the same thing.
- Linux Cross Reference and Google are great aids.

# First Step … Get a Handle to Sysfs

- `struct device * my_dev;`

```
my_dev = root_device_register( "my_folder" );
```

  - Creates a folder: `/sys/devices/my_folder/`
  - Returns the device struct pointer, or NULL on error

- When You're Done …
  - `void root_device_unregister`
  - `( struct device * my_dev);`

# Using a `Struct Device`

- Not really that useful.
- Most Useful Field: Kobject Struct
  - `struct kobject * ko = my_dev->kobj;`

- Each folder has an associated `kobject` struct.
  - You'll need the `struct kobject *` to add subdirectories, add files, etc.

# Creating Subdirectories

- Making a new folder:

```
struct kobject * subdir =
  kobject_create_and_add(
    char * name,
    struct kobject * parent );
```

  - Returns NULL on error.

- Free it when you're done:
  - `void kobject_put( struct kobject * subdir );`

# Example

```
struct device * my_dev;
struct kobject *root, *s1, *s2;

// Create /sys/devices/my_folder
my_dev = root_device_register( “my_folder” );
root = my_dev->kobj;

// Create /sys/devices/my_folder/subdir1/
s1 = kobject_create_and_add( “subdir1”, root );

// Create /sys/devices/my_folder/subdir1/subdir2/
s2 = kobject_create_and_add( “subdir2”, s1 );
```

# Bring on the Structs …

- Each file is associated with a set of structs.

```
struct attribute {
    char *              name;  // Name of file
    struct module *     owner;
    umode_t             mode;  // Octal
permissions
};
```

- Initialize the name and mode fields

```
struct attribute attr;
attr.name = "attr1";
attr.mode = 0666; // rw-rw-rw
```

# Layer II: `device_attribute`

- The `struct attribute` is a field inside of a larger `struct`

```
struct device_attribute {
    struct attribute attr;
    ssize_t (*show)
        ( struct device * dev,
          struct device_attribute *attr,
          char * buf );
    ssize_t (*store)
        (stuct device * dev,
         struct device_attribute * arrr,
         const char * buf,
         size_t count );
};
```

# That's Weird Syntax …

- It's C's way of saying "Function Pointer"
  - The `show` function is called when your sysfs file is read.
    - Place the data you want to pass back into the buffer.
    - Return the number of bytes placed into the buffer.
    - Buffer is `PAGE_SIZE` bytes.
    - Use `scnprintf()` instead of `snprintf()`
      - https://lwn.net/Articles/69419/
  - More on the first two arguments later…

# That's Weird Syntax … Part 2

- The `store` function is called when your sysfs file is written to.
  - The third argument is a buffer containing the data.
  - The fourth argument is the number of bytes of data in the buffer.
  - Return the number of bytes consumed – normally, just return the 4th argument to say you processed all the data.
  - More on the first two arguments later…

# Back to Creating Files

- If your file will be read, create a function

```
ssize_t mydev_do_read
    ( struct device * dev,
      struct device_attribute * attr,
      char * buf );
```

- If your file will be written to, create a function

```
ssize_t mydev_do_write
      ( struct device * dev,
        struct device_attribute * attr,
        const char * buf,
        size_t count );
```

# Creating Files

- Then, create and initialize `device_attribute` struct
  - Must be either a global variable (ideally, static), or dynamically allocated.

```
// Global declaration
static struct device_attribute my_dev_attr;

// Later on in the file
my_dev_attr.attr.name = ”my_file";
my_dev_attr.attr.mode = 0444;          // Read Only
my_dev_attr.show  = mydev_do_read;     // Read Function
my_dev_attr.store = NULL;              // No W function
```

# Adding 1 File

- Once we have the `device_attribute` structure, we can add the file to the folder.

```
int sysfs_add_file(
    struct kobject * folder,
    const struct attribute * attr );
```

  - Kobject * and Attribute * are passed as first two arguments to read and write functions.
    - A handy way to identify which folder was involved if you have many folders with the same files in each one.
    - Argument is a `struct device *,` but the pointers are the same
      - Not sure why this is

# Continuing the Examples …

```
static struct device_attribute my_dev_attr;
struct kobject * s1;
if(PTR_ERR(sysfs_create_file( s1, &my_dev_attr.attr ) ) )
{
  // Handle Error

}
```

# Cleaning Up: Removing One File

```
void sysfs_remove_file(
   struct kobject * folder,
   const struct attribute * attr
);
```

- Again continuing the examples

```
sysfs_remove_file( s1, &my_dev_attr.attr);
```

# Adding Multiple Files at Once

- Attributes can be bundled together.
- First, create a **`null`** terminated array of
- `struct attribute * 's`
- Example:

```
struct attribute * bundle[ 4 ];
struct device_attribute da1;
struct device_attribute da2;
struct device_attribute da3;
// Initialize each device attribute
bundle[0] = &da1.attr;
bundle[1] = &da2.attr;
bundle[2] = &da3.attr;
bundle[3] = NULL;
```

# struct attribute_group

- An attribute group stores the null terminated array of attribute pointers.

```
struct attribute_group {
    const char          *name;
    umode_t              (*is_visible)(struct kobject *,
                              struct attribute *, int);
    struct attribute    **attrs;
    struct bin_attribute  **bin_attrs;
};
```

- I only needed to set the `attrs` field(?)

```
struct attribute_group my_attr_grp;
my_attr_grp.attrs = bundle;
```

# Registering a Group of Attributes

```
int sysfs_create_group(
  struct kobject * folder,
  const struct attribute_group * group
);
```

  - Returns 0 on success; non-zero on failure

- Continuing the example. This code creates a file for each attribute in the array, `bundle`, pointed to from `my_attr_grp`, inside the folder corresponding to `s1`.

```
struct attribute_group my_attr_grp;
struct kobject *s1;
if( sysfs_create_group( s1, &my_attr_grp ) )
{
  // Handle Error
}
```

# Cleaning up a Group

```
void sysfs_remove_group(
  struct kobject * folder,
  struct attribute_group group
);
```

```
struct attribute_group my_attr_grp;
struct kobject *s1;
sysfs_remove_group( s1, &my_attr_grp );
```

# Other Useful `sysfs` Functions

To convert strings to integers at the kernel level

```
u32 number =
  simple_strtol(
    char * buffer,
    char ** end
    unsigned int base
  );
```

- This is particularly useful in the write functions.
- First argument is a buffer containing the string.
- Second argument is pointer that will be set to point to the end of the processed number.
- Third argument is the base to use (probably 10)