Kernel USB Gadget Configfs Interface

Matt Porter
Linaro

Overview

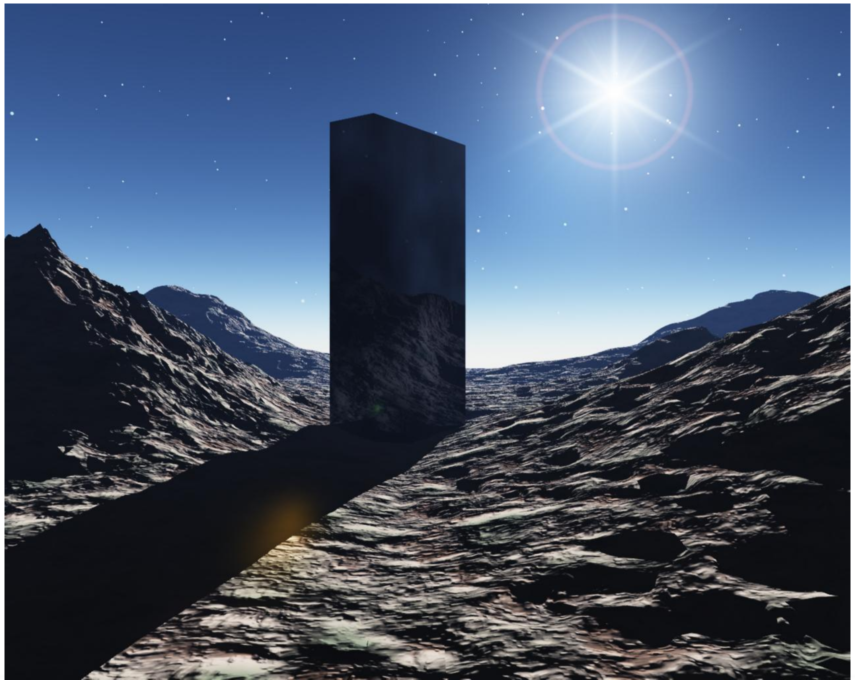
- Prereqs: understand USB
- Linux USB Terminology
- Brief history of USB gadget subsystem
- Other filesystem-based gadget interfaces
- Using USB gadget configfs
- libusbg
- Demo

Linux USB Terminology

- USB host driver - The USB Host Controller driver
- USB device driver - USB host-resident driver that supports a USB peripheral
- UDC driver - USB Device Controller driver
- Gadget driver - Driver implementing peripheral functionality

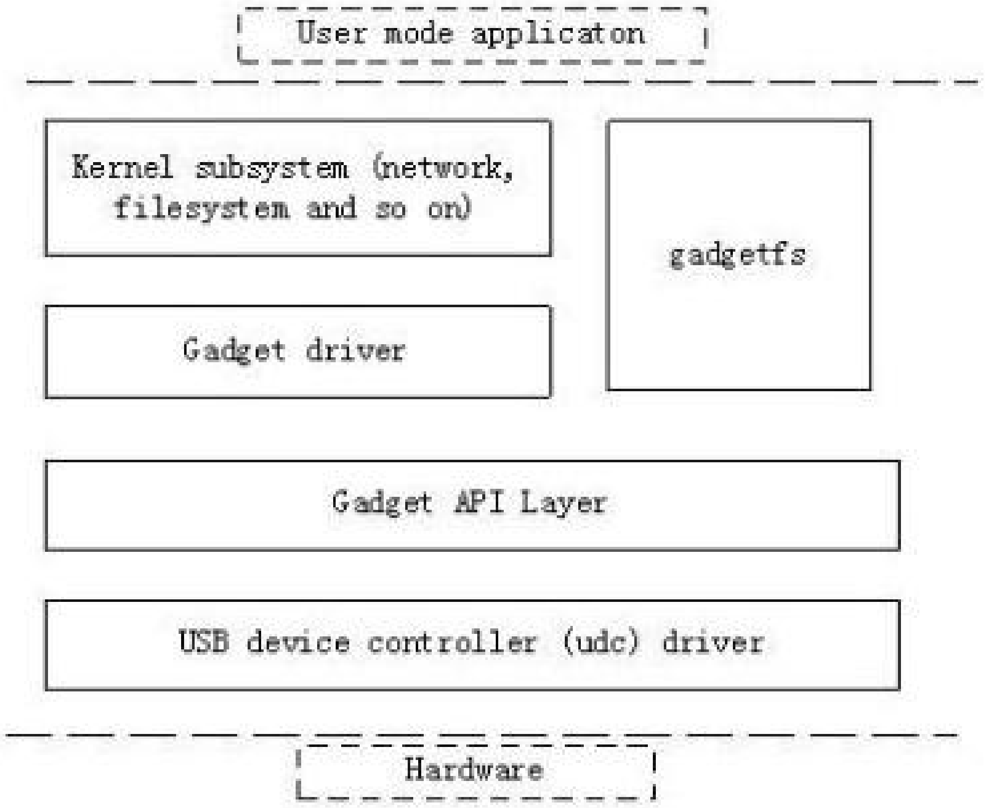
Linux USB Gadget History

- David Brownell introduces gadget framework in early 2003
  - the community endlessly debates the term “gadget”
  - supports only monolithic gadget drivers
  - g_zero and g_ether
  - a few usb device controller drivers



Linux USB Gadget History

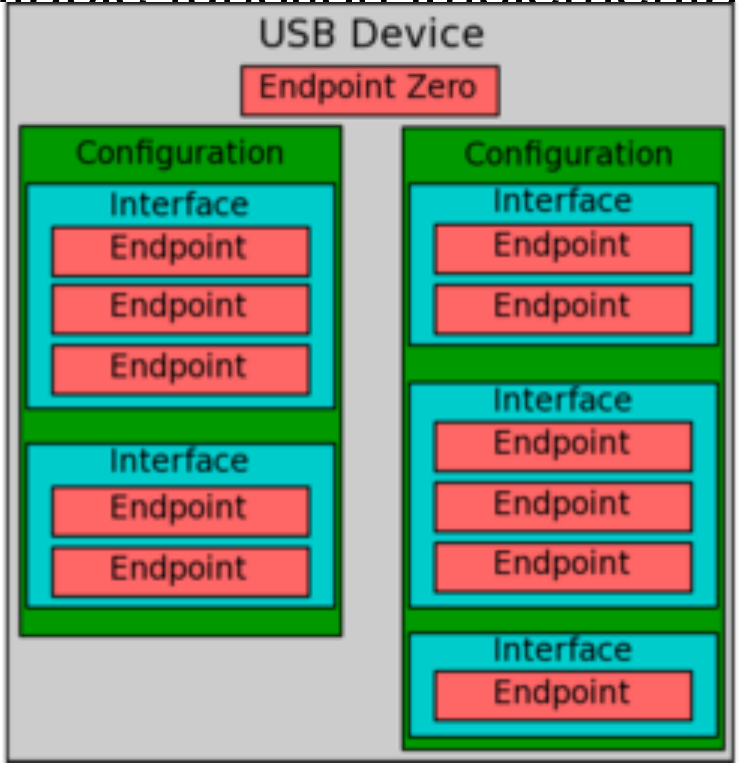
- gadgetfs introduced in late 2003
  - enables userspace gadget drivers
  - MTP/PTP is a common use case


User mode applicaton
Kernel subsystem (network, filesystem and so on)
gadgetfs
Gadget driver
Gadget API Layer
USB device controller (udc) driver
Hardware

http://img.ph.126.net/QzNmkfawYfnYDlg3Inza9A==/2856408063661382344.jpg

Linux USB Gadget History

- composite framework added in 2008
  - enables multi-function (or USB composite) gadget drivers
  - existing gadget drivers slowly moved over to compositable function implementations



http://lwn.net/Articles/395712/

Linux USB Gadget History

- FunctionFS added in 2010
  - compositable version of gadgetfs
  - now userspace gadget functions can be combined with kernel gadget functions in a composite gadget
  - e.g. mass storage (kernel) + MTP (via FunctionFS)

But still… something is missing...

Flexibility!

We are still stuck creating custom kernel modules to glue N instances of M functions together for our unique use cases

USB Gadget ConfigFS

- Our hero finally arrives in 3.11
- What is it?
  - A userspace API for creation of arbitrary USB composite devices using reusable kernel gadget function drivers.
  - Supports all major existing gadget functions except FunctionFS and mass storage in 3.11
  - 3.13 added conversion of FunctionFS and mass storage

Huh? Explain all these filesystems!

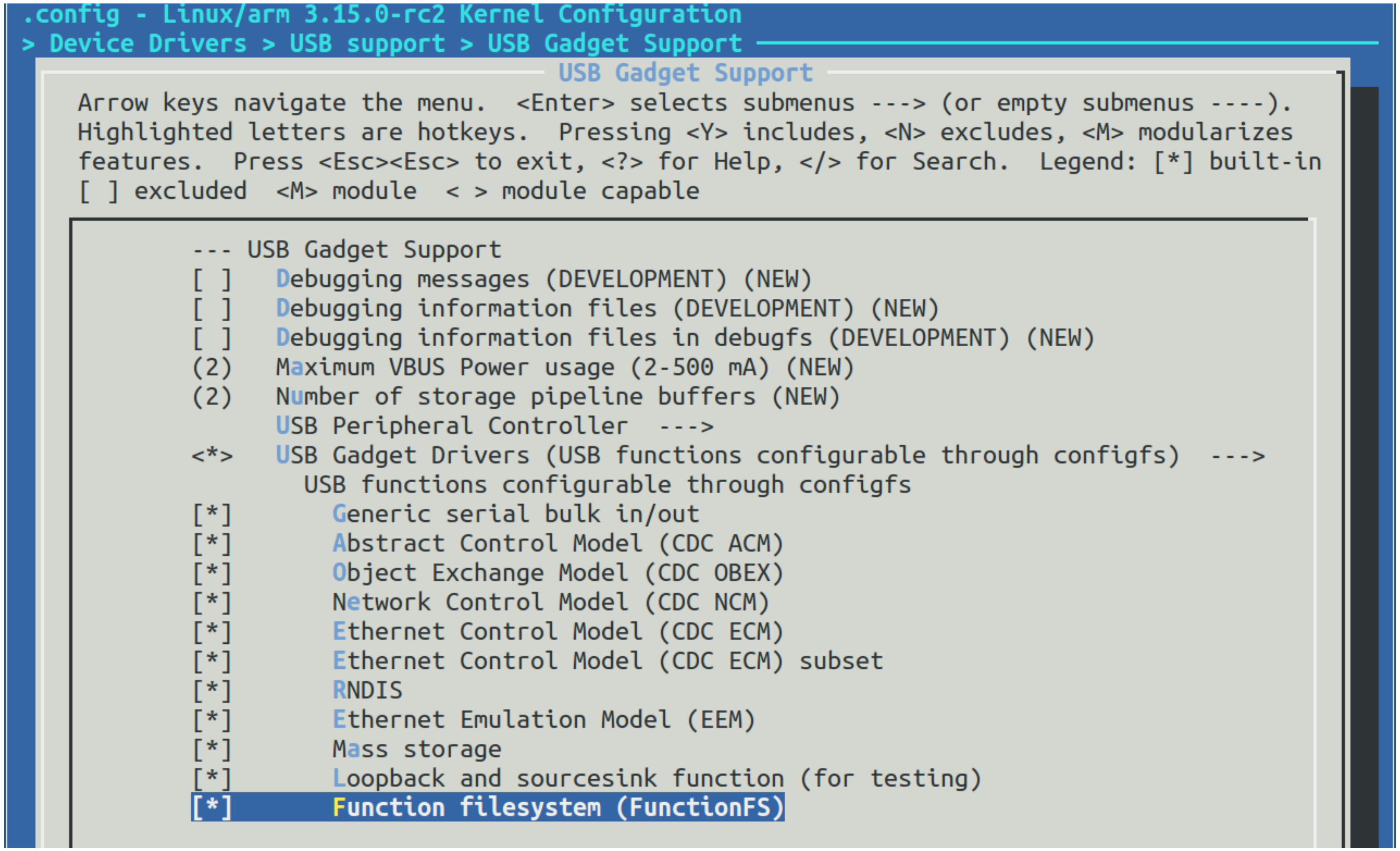
- Review
  - GadgetFS - original monolithic kernel driver that provides an interface to implement userspace gadget drivers
  - FunctionFS - rewrite of GadgetFS to support userspace gadget functions that can be combined into a USB composite gadget.
  - USB Gadget ConfigFS - interface that allows definition of arbitrary functions and configurations to define an application specific USB composite device from userspace.

But why use configfs?

- sysfs versus configfs
  - sysfs exposes kernel created objects to userspace
  - configfs allows userspace instantiation of kernel objects
- configfs is the appropriate model for creation of gadget devices
  - create the gadget device and bind to a UDC driver from userspace

Enabling USB Gadget ConfigFS

Exact steps



```
.config - Linux/arm 3.15.0-rc2 Kernel Configuration
> Device Drivers > USB support > USB Gadget Support
                              USB Gadget Support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

        --- USB Gadget Support
        [ ]   Debugging messages (DEVELOPMENT) (NEW)
        [ ]   Debugging information files (DEVELOPMENT) (NEW)
        [ ]   Debugging information files in debugfs (DEVELOPMENT) (NEW)
        (2)   Maximum VBUS Power usage (2-500 mA) (NEW)
        (2)   Number of storage pipeline buffers (NEW)
              USB Peripheral Controller  --->
        <*>   USB Gadget Drivers (USB functions configurable through configfs)  --->
                USB functions configurable through configfs
        [*]       Generic serial bulk in/out
        [*]       Abstract Control Model (CDC ACM)
        [*]       Object Exchange Model (CDC OBEX)
        [*]       Network Control Model (CDC NCM)
        [*]       Ethernet Control Model (CDC ECM)
        [*]       Ethernet Control Model (CDC ECM) subset
        [*]       RNDIS
        [*]       Ethernet Emulation Model (EEM)
        [*]       Mass storage
        [*]       Loopback and sourcesink function (for testing)
        [*]       Function filesystem (FunctionFS)
```

Mounting USB Gadget ConfigFS

Exact steps

```
# mount -t configfs none /sys/kernel/config
# cd /sys/kernel/config/
# ls
usb_gadget
# cd usb_gadget
```

If USB Gadget configfs support is enabled we'll have a usb_gadget subdirectory present

Create 2xACM + ECM Gadget

Exact steps

```
# mkdir g1
# cd g1
# ls
UDC            bDeviceProtocol  bMaxPacketSize0  bcdUSB   functions
idVendor
bDeviceClass   bDeviceSubClass  bcdDevice        configs  idProduct  strings
```

By creating the g1 directory, we've instantiated a new gadget device template to fill in.

Create 2xACM + ECM Gadget

Exact steps

```
# echo "0x1d6b" > idVendor
# echo "0x0104" > idProduct

# mkdir strings/0x409
# ls strings/0x409/
manufacturer  product  serialnumber

# echo "0123456789" > strings/0x409/serialnumber
# echo "Foo Inc." > strings/0x409/manufacturer
# echo "Bar Gadget" > strings/0x409/product
```

Write in our vendor/product IDs

Instantiate English language strings

Write in our serial number, manufacturer, and product descriptor strings

Create 2xACM + ECM Gadget

Exact steps

```
# mkdir functions/acm.GS0
# mkdir functions/acm.GS1
# mkdir functions/ecm.usb0
```

Create function instances. Note that multiple function instances of the same type must have a unique extension

Create 2xACM + ECM Gadget

Exact steps

```
# mkdir configs/c.1
# ls configs/c.1
MaxPower  bmAttributes  strings
```

Create a configuration instance

```
# mkdir configs/c.1/strings/0x409
# ls configs/c.1/strings/0x409/
configuration
# echo "CDC 2xACM+ECM" > configs/c.1/strings/0x409/configuration
```

Create English language strings and write in a description for this device configuration

```
# ln -s functions/acm.GS0 configs/c.1
# ln -s functions/acm.GS1 configs/c.1
# ln -s functions/ecm.usb0 configs/c.1
```

Bind each of our function instances to this configuration

Create 2xACM + ECM Gadget

Exact steps

```
# ls /sys/class/udc/
3f120000.usb
```

Verify which UDC drivers are available

```
# echo "3f120000.usb" > UDC
```

Attach the created gadget device to our UDC driver.

libusbg

- Library providing C API to USB Gadget Configfs
  - Supports creation and removal of gadgets
  - Full API docs at http://libusbg.github.io/
  - Source at git://github.com/libusbg/libusbg.git
- Status
  - Patch review conducted on linux-usb-devel list
  - Starting to gain major contributions in cleanups and API improvements
    - Major contributions have come from Samsung, in particular, Krzysztof Opasiak

libusbg-based 2xACM + ECM Gadget

```
 usbg_gadget_strs g_strs = {
                "0123456789", /* Serial number */
                "Foo Inc.", /* Manufacturer */
                "Bar Gadget" /* Product string */
};

usbg_config_strs c_strs = {
                "CDC 2xACM+ECM"
};

usbg_init("/sys/kernel/config", &s);
usbg_create_gadget(s, "g1", &g_attrs, &g_strs, &g);
usbg_create_function(g, F_ACM, "GS0", NULL, &f_acm0);
usbg_create_function(g, F_ACM, "GS1", NULL, &f_acm1);
usbg_create_function(g, F_ECM, "usb0", NULL, &f_ecm);
 usbg_create_config(g, 1, "The only one", NULL, &c_strs, &c);
usbg_add_config_function(c, "acm.GS0", f_acm0);
usbg_add_config_function(c, "acm.GS1", f_acm1);
usbg_add_config_function(c, "ecm.usb0", f_ecm);
usbg_enable_gadget(g, DEFAULT_UDC);
usbg_cleanup(s);
```

Use function enums let the compiler catch our typos

Default UDC will just use the first or only one listed in sysfs

Demo