Do More With Less

On Driver-less Interfacing with Embedded Devices

Peter Korsgaard <peter@korsgaard.com>
Driver-less Interfacing?
Interfacing without having to install any custom SW on PC
Embedded == Custom Stuff
Embedded == Custom Stuff
But Why?
.. Its easy
.. Its easy .. At First
Time Pressure
Flexibility
I'm Special
Does It Need To Be So?

- While these are sometimes valid concerns, often standard interfaces can be used instead
- Work done with Fabien Chouteau
Primer
Host

Devices
Device
  Configuration (mode)
    Interface (functionality)
    Endpoint (pipe)
A device with multiple interfaces is called a composite device.
Functions can implement

- Specific vendor protocols (custom)
- USB class protocols (standard)
Class protocols most interesting as OS'es have **built in** drivers

- HID (keyboard/mouse/..)
- Storage (Hard drivers, USB sticks)
- Audio (headsets, speakers)
- Video (webcams)
- ..
Linux USB Gadget Stack

- User Space
- Other Kernel Subsystems
- USB Function Driver
- USB Function Driver
- Composite
- USB Peripheral Controller Driver
Examples
Function Keys

Display with function keys used to control PC
Issues

Historically interfaced to PC using custom serial protocol

- Custom PC SW needed
- Support issues
- Not usable during BIOS/BOOT
- New PCs lack serial
Solution

Emulate USB HID keyboard
Human Interface Devices Class

- HID descriptors to specify device type and supported report (message) formats
- Reports to device to set properties (E.G LEDs on keyboard)
  - On control endpoint
- Reports from device to report changes (E.G. key presses / releases)
  - On interrupt endpoint
HID Keyboard (Boot) Protocol

- 1 byte reports to device
  - Bitmask of LED states (numlock, capslock, ..)
- 8 byte reports from device
  - Modifier key mask (alt, ctrl, ..)
  - and currently pressed keys

See HID usage tables for key code definitions
Implementation

- HID gadget function driver
- **Mainline** since 2.6.35
- Split kernel / user space implementation
  - HID descriptor handling in kernel,
  - `/dev/hidgX` character device to get/set HID reports
- See [Documentation/usb/gadget_hid.txt](Documentation/usb/gadget_hid.txt) for details
Kernel Side

- Platform device in platform code defining HID device descriptor(s)
- Can emulate as many devices as controller has endpoints
- g_hid USB gadget driver
static struct hidg_func_descriptor hid_data = {
    .subclass = 0, /* No subclass */
    .protocol = 1, /* Keyboard */
    .report_length = 8,
    .report_desc_length = 63,
    .report_desc = {
        0x05, 0x01, /* USAGE_PAGE (Generic Desktop) */
        0x09, 0x06, /* USAGE (Keyboard) */
        0xa1, 0x01, /* COLLECTION (Application) */
        ...
    }
};

static struct platform_device hid = {
    .name = "hidg",
    .id = 0,
    .num_resources = 0,
    .resource = 0,
    .dev.platform_data = &hid_data,
};

platform_device_register(&hid);
Read/write to /dev/hidgX

E.G. To send 'a':

```
  echo -en '\0\0\4\0\0\0\0\0' >/dev/hidg0
  echo -en '\0\0\0\0\0\0\0\0\0' >/dev/hidg0
```
Demo
Similar Setups

Yubico UbiKey one-time password generator

ThinkGeek Phantom Keystroker
Potential Pitfalls

Key codes in HID reports are scancodes Corresponding key depends on PC keyboard layout
Example 2
Data Transfers

Firmware Upgrades through USB
Issues

Historically using custom serial protocol
- Custom PC SW needed
- Support issues
- New PCs lack serial
Solution

Emulate USB memory stick

Alternatively access USB stick if host port available.
Pitfalls

- Mass Storage == Block device
- Filesystems / OSes don't support **concurrent** access
- Need to detect when it is **safe** to access device
Use Case

- Provide virtual drive where firmware upgrade can be copied to
- Perform upgrade when unplugged
Use Case

- Provide virtual drive where firmware upgrade can be copied to
- Perform upgrade when unplugged
Use Case

- Provide virtual drive where firmware upgrade can be copied to
- Perform upgrade when unplugged
Implementation

- File storage gadget function driver in kernel
- Userspace notification on unplug / eject
- **Mainline** since 2.6.35
- Sysfs attributes:
  - `/sys/<gadget>/suspended`
  - `/sys/<gadget>/lunX/file`
Implementation

- User space program that on unplug / eject:
  - Ejects file
  - Loopback mounts filesystem
  - Inspects it for interesting files
  - Recreates file system
  - Adds file to file storage driver
Implementation

- File system could simply be a pregenerated template
  - Prepopulated with any needed help/documentation files
- Stored on local storage or RAM (tmpfs)
  - If tmpfs, sparse file interesting
- FAT table / help file << filesystem size
Demo

- Check for image files
- Show on framebuffer
Similar Setups

• Same approach could be used to transfer data from device
• Several 3G modems have Windows drivers on emulated USB drive
Alternatives

- Device Firmware Upgrade (DFU)
- Picture Transfer Protocol (PTP)
  - Gadgetfs implementation: http://git.denx.de/?p=ptp-gadget.git
- Media Transfer Protocol (MTP)
  - MeeGo implementation: http://wiki.meego.com/Buteo/MTP

- None are as generic or well supported
Web Interfaces

- Good Alternative to custom PC GUI Software
- Many open source libraries exists
  - JQuery UI (GUI Widgets)
  - Flot (Graphs)
- Modern AJAX is nice for embedded
  - Heavy processing on client side
Conclusion

• USB Class protocols can be (ab)used for driver-less interfacing
• Easiest PC SW support is NO SW
• Easy to integrate, in mainline
• Extends to lots of other areas
Thanks!

Questions?