Kernel USB API vs Userspace (libusb)

Vasily **anarsoul** Khoruzhick

email: **anarsoul@gmail.com**, *skype:* **anarsoul**

github.com/anarsoul

Template of (minimal) libusb app

- libusb_init()
- dev_handle = libusb_open_device_with_vid_pid(vid, pid)
- libusb_claim_interface(dev_handle, intf_num)
- libusb_set_interface_altsetting()
- libusb_alloc_transfer() ... libusb_fill_*_transfer() ... libusb_submit_transfer() ... completion callback ...
- libusb_close(dev_handle)
- libusb_exit()

Template of (simple) kernel driver for USB device

- USB ID compatibility table (possible to match device class also)
- driver probe function
- usb_driver_claim_interface()
- usb_set_interface()
- usb_alloc_urb() ... usb_fill_*_urb() ... usb_submit_urb() ... completion callback ... usb_free_urb()
- driver removal function

Generic Kernel vs Userspace differences

- No one cleanups after you in Kernel
- Dereferenced NULL pointer? -> reboot
- Corrupted some memory of other driver? -> reboot
- Atomic/non-atomic context

Entry point

- Userspace: should obtain device handle by itself (libusb_open_device_with_vid_pid())
- Kernel: contains USB ID compatibility table, device handle is passed by driver core

Async API: libusb_transfer vs usb_urb

- Represents single USB transfer
- Contains source & destination (EP num, direction, pointer to buffer and its size)
- Completion callback
  - In kernel completion callback is called from atomic context

Async API libusb_submit_transfer() vs usb_urb_submit()

- Almost no difference - both submit asynchronous transfer

Sync API: libusb_{control,bulk,interrupt}_transfer vs usb_{control,bulk,interrupt}_msg

- Kernel: can't use sync API in atomic context
- Kernel: userspace: it's convenient, but usually not a good idea to use synchronous API

Userspace: Pros and Cons

- Pros:
  - Good for prototyping, decent development speed
  - Easy to debug
  - Convenient synchronous API
- Cons:
  - Performance and response latency is worse

Kernel: Pros and Cons

- Pros:
  - Decent performance and response latency
- Cons:
  - Development speed is slower
  - Not so easy to debug
  - One should always keep in mind that it's a kernel:
    - ★ Can't sleep in completion callback
    - ★ Can't use synchronous API in atomic context
    - ★ Be carefull with memory and other resources
    - ★ Async stuff is complicated, sync stuff in kernel is evil.

The End

Thanks! Questions?