

# The new USB stack in FreeBSD 8/9

by

Hans Petter Selasky (hselasky @)

Master in Technology, in Information and Communication Technology,  
at Agder University College in Norway, Faculty of Engineering and Science

MeetBSD, Krakow, Poland

July 2010

# Quick overview

- USB introduction
- USB data transfers
- USB stack and SMP
- USB callback threads
- USB polled mode
- USB in userspace
- . . .
- History and lessons learned

# What is USB?

- USB is a 2-wire half-duplex asymmetric protocol designed to connect an external peripheral to your computer. Two additional wires are used to supply electricity (+5.0V 0..500mA) to your device.
- USB has 4 quality of services categories for data transfers:
  - ISOCHRONOUS (fixed rate and no retransmit)
  - INTERRUPT (fixed rate and retransmit)
  - CONTROL (non-fixed rate and retransmit)
  - BULK (non-fixed rate and retransmit)

# What is USB?

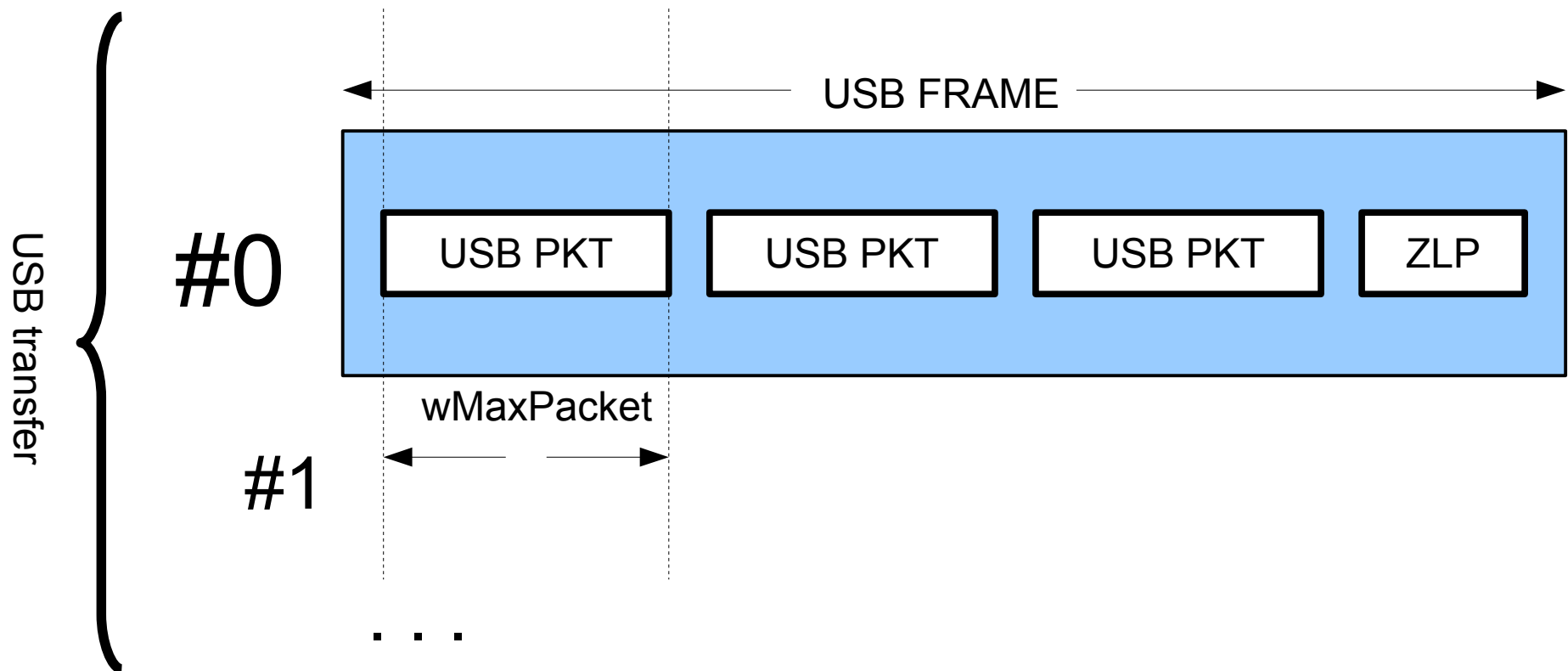
- Read more about USB at:
  - <http://www.usb.org>

# USB data transfer management

- Symmetric API for Host and Device mode and supported USB transfer types.
  - Blocking calls
    - `usb_transfer_setup()`
      - pre-allocation of resources yields
        - reduced execution latency
        - reduced CPU usage
    - `usb_transfer_unsetup()`
    - `usb_transfer_drain()`
  - Non-blocking calls
    - `usb_transfer_start()`
    - `usb_transfer_stop()`

# USB data transfer management

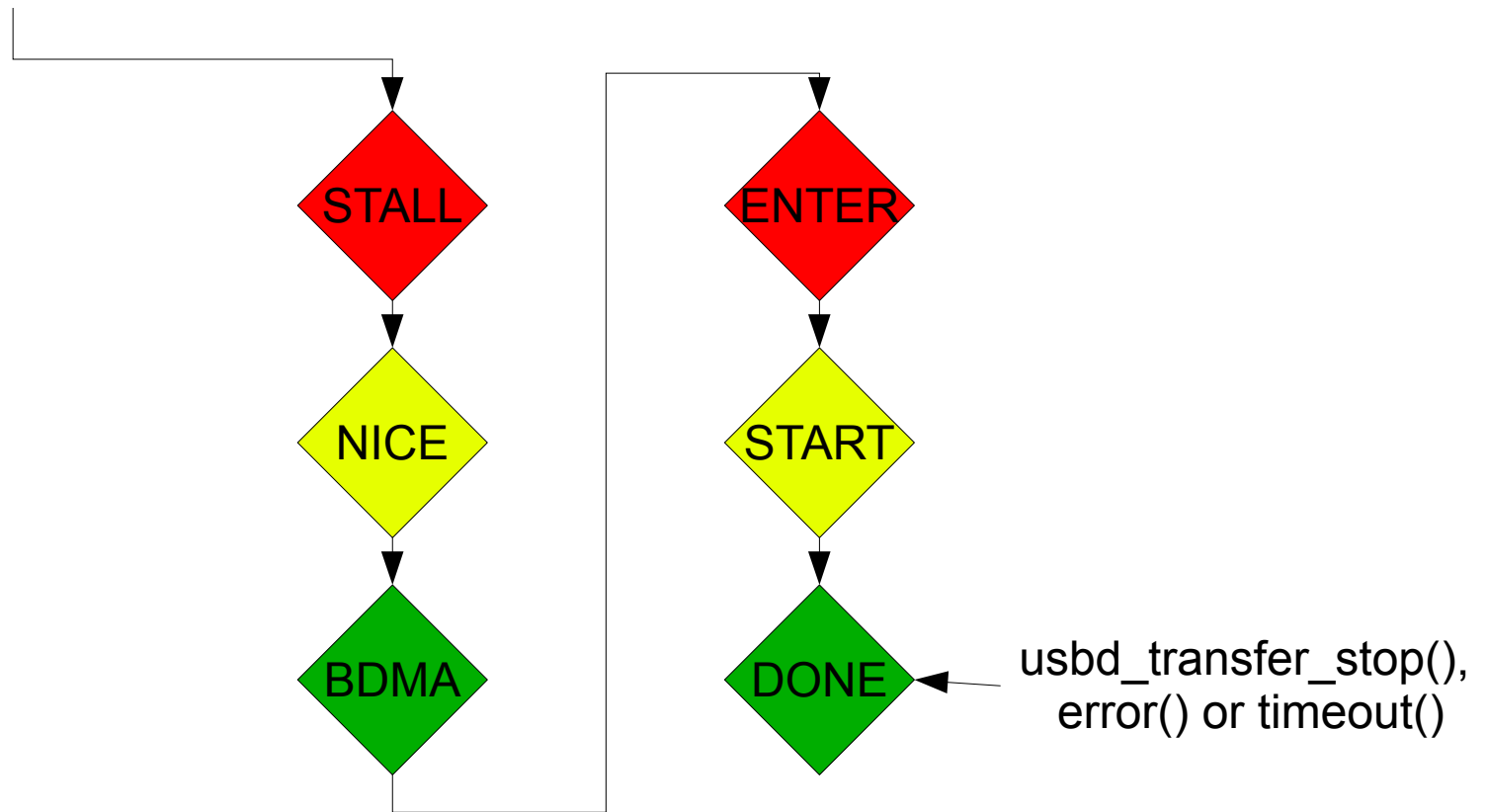
- USB transfers are now I/O vectored
- Multiple USB frames can be transmitted in a single USB transfer having a pointer and length.



# USB data transfer management

- From start to stop

usb\_transfer\_submit()

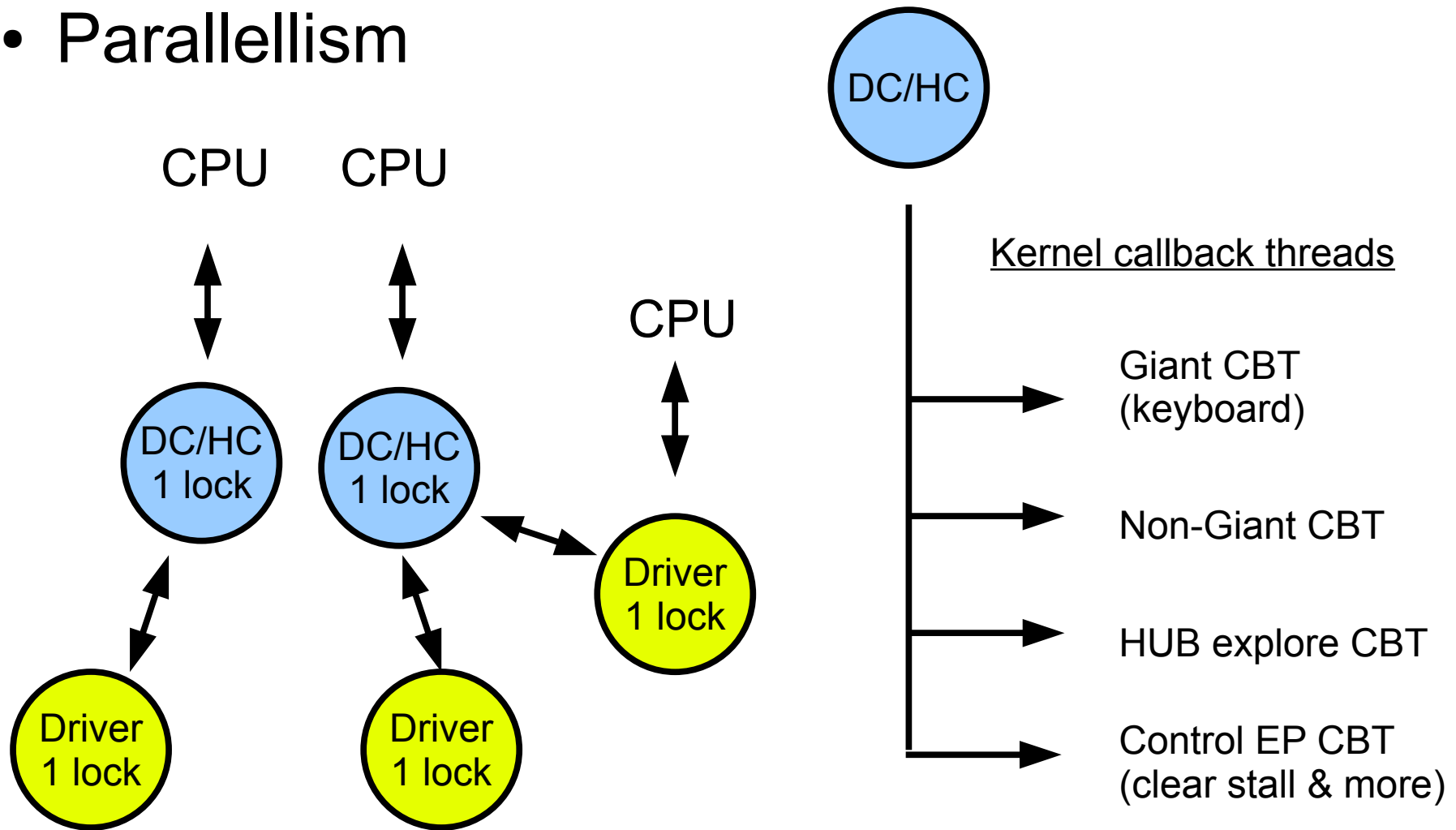


# USB control transfers

- Supports up to 64KByte of payload re-using the same 1KByte buffer. A single control transaction can be split across multiple USB transfers.
- Supports transferring data directly to/from user-space using copyin/copyout to a fixed-size kernel buffer. This saves a kernel malloc and free per control request and gives a significant latency reduction.
- Takes a mutex argument which is passed to `mtx_sleep()` and the alike.

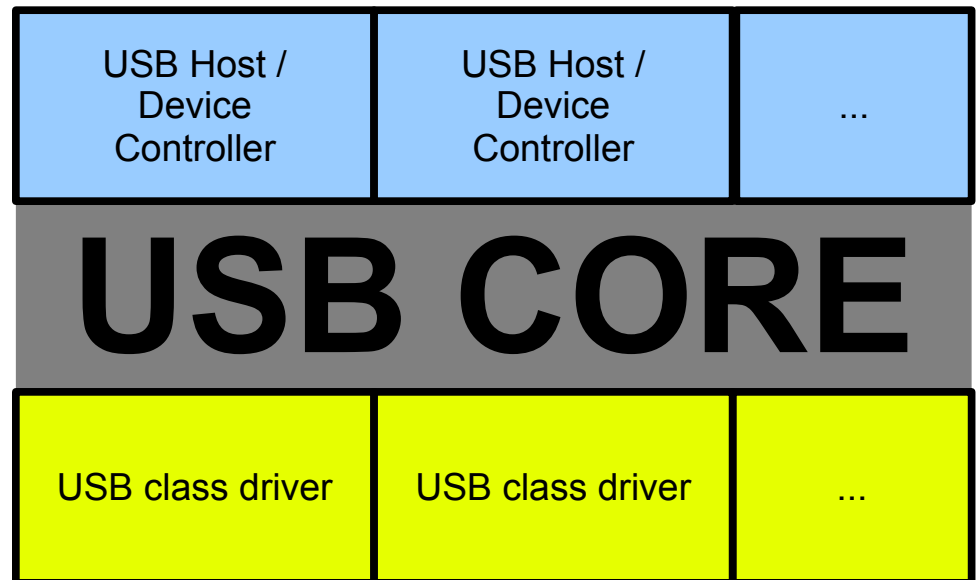
# USB stack and SMP

- Parallelism



# USB stack and SMP

- Vertically – two mutexes
  - USB class driver mutex (one per class driver)
  - USB controller driver mutex (one per controller)
  - 2 locks = 4 lock combinations



# USB stack and SMP

- The USB class driver lock must be locked before starting or stopping any data transfers
- The USB class driver lock is automatically locked before the data transfer callback is called.
- Reduces the need for mutex operations in drivers.
- Resolves corner cases. What happens if ...
- **Idea:** Running code atomically with regard to a mutex and integrating locking and unlocking into the APIs.

# USB stack and SMP

- The USB transfer structure is always referred to by a pointer. The USB transfer methods are NULL safe.

```
mtx_lock(&sc->sc_mtx);  
usbd_transfer_start(sc->sc_xfer[0]);  
mtx_unlock(&sc->sc_mtx);
```

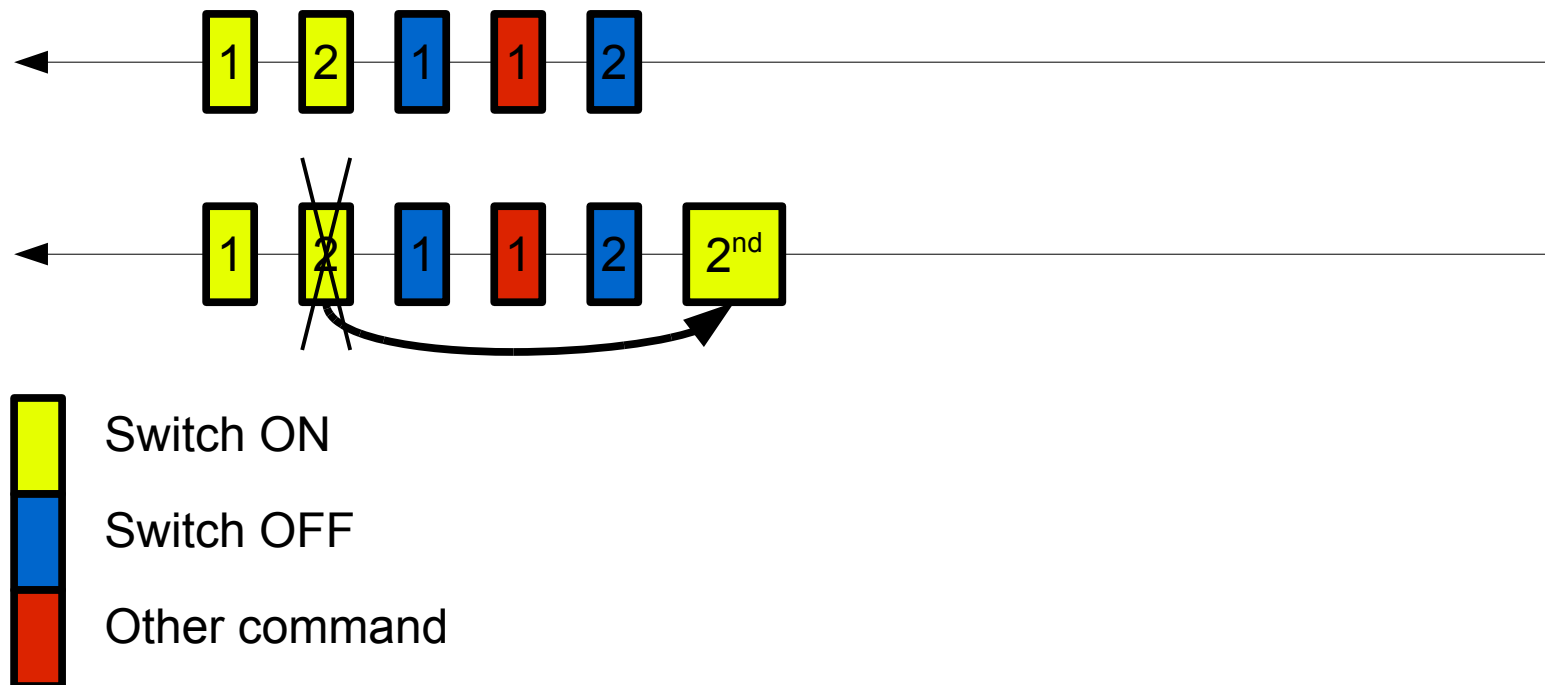
This ensures atomicity when tearing down the transfers. Either the USB transfer pointer is set and the transfer is started or the USB transfer pointer is not set and the transfer is not started.

# USB stack and SMP

- Synchronous USB control transfers take a mutex argument for sake of convenience that will be exited and entered before return if blocking operation is required.

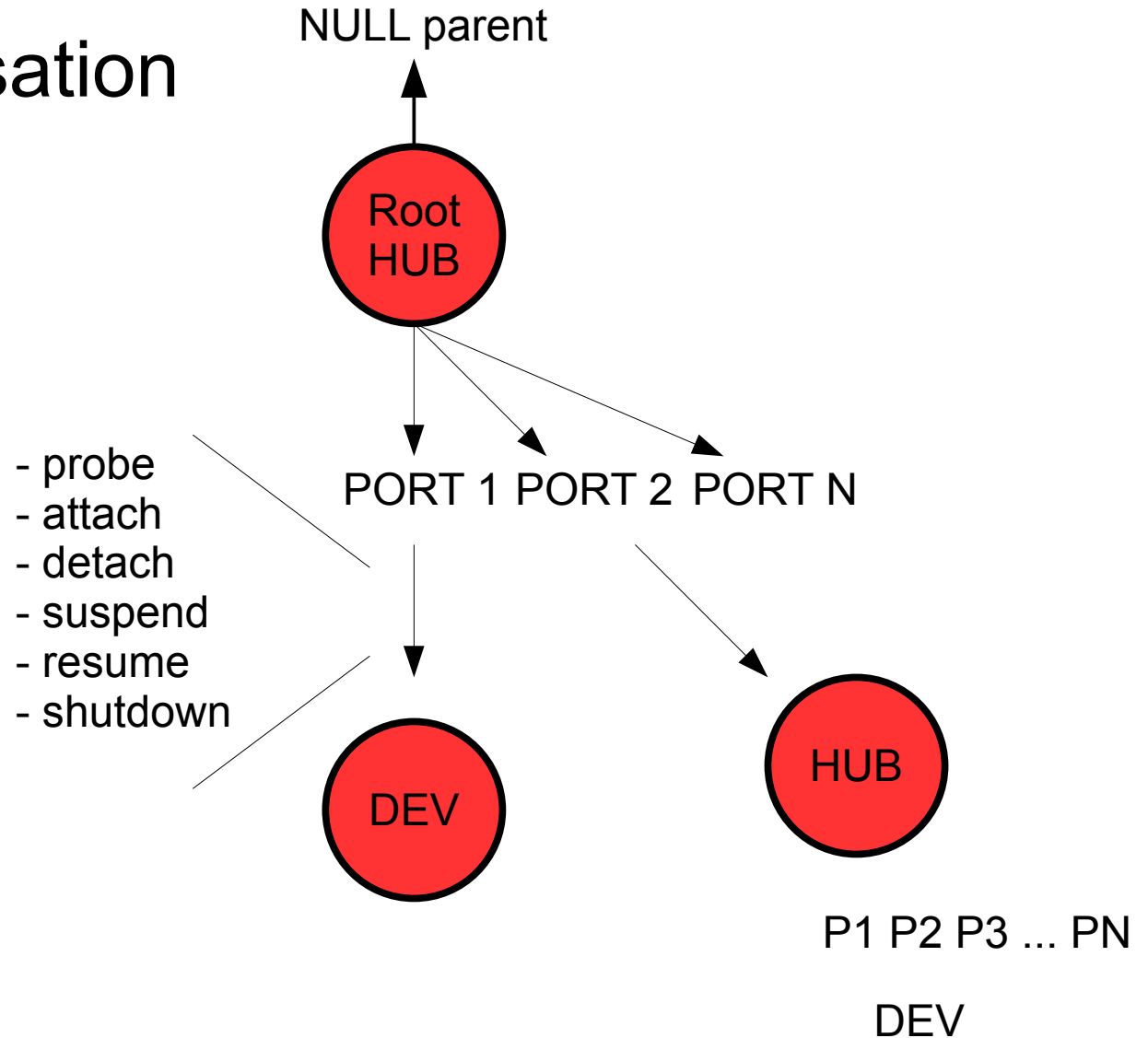
# USB callback threads

- Almost like a task queue, except two queue entries per task.
- Rule: Last queued is last executed. A task queue does not guarantee this.



# USB explore callback thread

- Device organisation



# USB explore callback thread

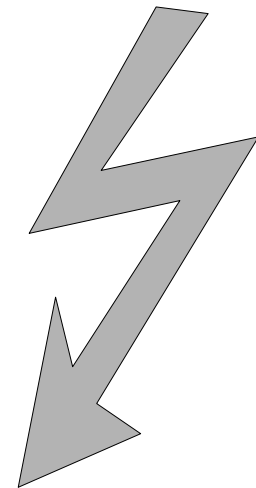
- Power save
  - Integrated in USB HUB code
  - Seamless with regard to USB transfers
  - 30 second suspend delay
    - Turned off by default due to many broken USB devices.
    - `usbconfig -d x.y power_save`

# USB explore callback thread

- Port event
  - Status change
- Port status
  - Suspend / Resume
  - Enabled / Disabled
  - Host / Device mode (new BSD specific)
  - Cable transmit speed

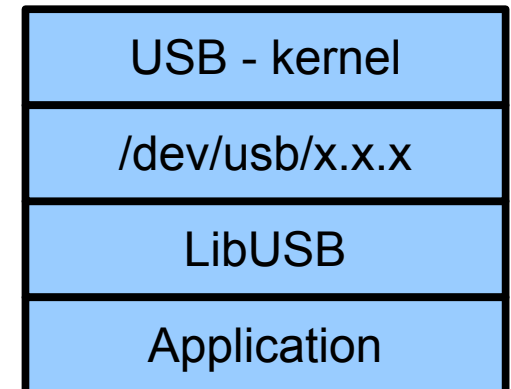
# USB stack and polled mode

- Only USB data transfers can be polled. Enumeration requires thread context.
- Allows you to use the USB keyboard or USB dump device in KDB, after a panic or during early boot.



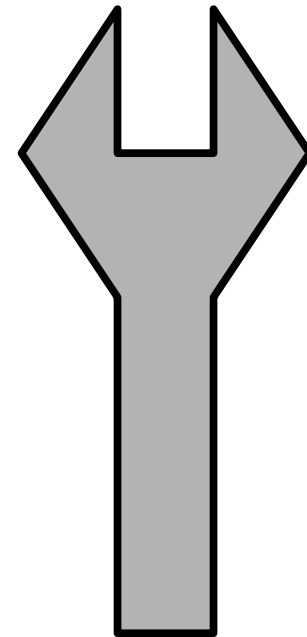
# USB and userspace

- LibUSB v0.1 API
- LibUSB v1.0 API
- LibUSB v2.0 (BSD specific API)
- Most of the functionality provided by the kernel USB API is exported to userspace via LibUSB
- Legacy endpoint nodes



# USB and userspace

- usbconfig utility – the swiss army knife of USB
  - Select USB configuration
  - Dump USB descriptors
  - Manage USB quirks
  - Perform USB control requests
  - Select USB power mode
  - Sits on top of LibUSB v2.0.



# USB Mass Storage Tester

- Some USB devices provide autoinstall software mostly for closed source operating systems. These are detected and skipped when possible.
- For example: 3G wireless dongles

# USB throughput and performance

- Nothing more and nothing less than what is expected.
- FullSpeed USB (v1.0)
  - 0.8-1.0MByte/second
  - 1000 IRQ/s
- HighSpeed USB (v2.0)
  - 30-35MByte/second (depends and Host and Device)
  - 8000 IRQ/s

# USB throughput and performance

- USB HighSpeed test, reading data from the same external USB memory stick:
  - Command used  
`dd if=/dev/rdisk1 of=/dev/null bs=65536 count=1000`
  - Mac OS X 10.5.8 on MacBook Pro  
65536000 bytes transferred in 3.710055 secs (17664428 bytes/sec)
  - FreeBSD 9-current on MacBook Pro  
65536000 bytes transferred in 6.014830 secs (10895736 bytes/sec)  
The low throughput is because the Nvidia chipset used has the IAAD bug. See last test result for relevant comparison.
  - Ubuntu 9.04 on MacBook Pro  
65536000 bytes (66 MB) copied, 4.0388 s, 16.2 MB/s
  - FreeBSD 8-stable on Acer Travelmate (Intel chipset)  
65536000 bytes transferred in 3.563875 secs (18388973 bytes/sec)

# Security and reliability

- Extra time has been spent during development to ensure proper range checks and USB descriptor validation.
- Extra time has been spent during development to identify and resolve races during detach.  
For example in `/dev/usb/x.x.x`

# History and lessons learned

- Started due to missing HW driver support in FreeBSD in 1999 when I was 17 years old. It ended up with a brand new USB stack in 2009. The USB stack was mainly developed in my sparetime after doing my homework assignments when going to school for many years.
- I´m very thankful for the trust I´ve been given to make the new USB stack in FreeBSD.
- I hope that my work will be an inspiration to others for many years to come!

Questions ?