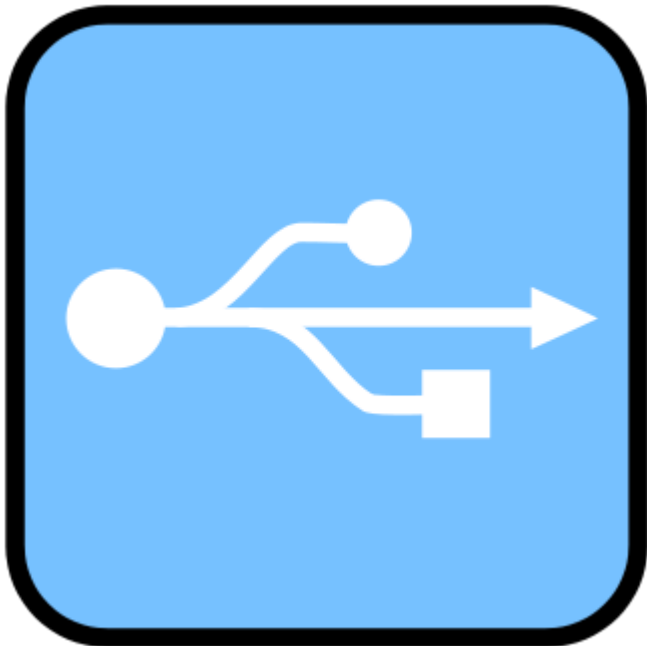


# USB und libUSB



Daniel Käfer

UnFUG, 1. Dezember 2011

# Inhalt

- 1. USB**
- 2. LibUSB**
- 3. Eigener Treiber entwickeln**

**USB**

# Was ist USB?

## Standard für:

- Kabel
- Stecker
- Signalübertragung
- Protokoll

# Architektur

- Bus
- Host
- Hub
- Plug and Play
- Nur poll / kein push

# Architektur - Host

- Kontrolliert die Verbindung
- Relative Teuer
- z.B. PC, Notebook

# Architektur - Endgerät

- Eigene Stromversorgung oder über den Bus
- Relative Preiswert
- Kann keine Nachricht an den Host senden

# Kommunikation

- Seriell
- Nur eine Richtung
- Beide Richtungen → 2 Endpunkte
- Timeslot Verfahren

# Transfer Type

- Isochroner Transfer
- Interrupt-Transfer
- Bulk-Transfer
- Control-Transfer

# Geräteklassen

- HID - Human Interface Device  
z.B. Tastatur, Maus
- MSC - Mass Storage Device  
z.B. Externe Festplatte, USB-Stick
- Audio, MTP, etc
  
- Vorteil: nur ein Treiber für viele Geräte

# Eigene Geräteklasse

- Nur Endpunkt 0 ist vorgegeben  
Für den Control Kanal
- Der Rest kann frei verwendet werden

# Vergleich mit IP-Netzwerk

USB	IP-Netzwerk
Endpunkt	Port
Bulk-Transfer	TCP
Isochroner Transfer	UDP
[[bus:]][devnum]	IP/Mac-Adresse

**libUSB**

# LibUSB – Übersicht (Teil 1)

- Verfügbar unter Linux, Darwin (Mac OS X) und Windows
- GNU Lesser General Public License
- Version 0.1
  - 2007
  - Wird noch oft eingesetzt
- Version 1.0
  - 2008

# LibUSB – Übersicht (Teil 2)

- User Space
- In c geschrieben

# Bindings

- Python – pyusb
- Perl – Device::USB
- Ruby – ruby-usb
- Java – libusbjava
- C# und .Net – libusbdotnet
- etc

# Beispiel – Gerät finden (Teil 1)

```
struct usb_bus *busses;
```

```
usb_init();
```

```
usb_find_busses();
```

```
usb_find_devices();
```

```
busses = usb_get_busses();
```

# Beispiel – Gerät finden (Teil 2)

```
struct usb_bus *bus;
for (bus = busses; bus; bus = bus->next) {
    struct usb_device *dev;
    for (dev = bus->devices; dev; dev = dev->next) {
        if (ID_VENDOR == dev->descriptor.idVendor &&
            ID_PRODUCT == dev->descriptor.idProduct) {
            // Device gefunden
        }
    }
}
```

# Beispiel – Verbinden

```
struct usb_dev_handle *handler;  
handler = usb_open(device);  
usb_detach_kernel_driver_np(handler, 0);  
usb_claim_interface(handler, 0);
```

# Beispiel – Daten senden

```
void send(char *bytes, int size) {  
    int requesttype = USB_TYPE_CLASS | USB_RECIP_INTERFACE;  
    int request = 0x09;  
    int value = 0x200;  
    int index = 0x00;  
    int timeout = 100;  
  
    usb_control_msg(handler, requesttype, request, value, index,  
bytes, size, timeout);  
}
```

# Alternative – Kernel space

- Treiber kann im Kernel implementiert werden
- Ähnlich Schnittstelle
- Installation vom Treiber ist aufwendiger
- Fehler haben größere Auswirkungen → Kernel Panic

**Treiber Entwickeln**

# Reverse Engineering

- „Original“ Treiber installieren
- Traffic mitschneiden
- Filtern nach Bus+Device oder Hersteller id + Produkt id

# Tools

- SniffUsb 2.0 for Windows

<http://www.pcausa.com/Utilities/UsbSnoop>

- Wireshark (USB nur unter Linux)

<http://www.wireshark.org>

# Setup

- VirtualBox
  - USB weiterreichen
  - Auf dem Host oder Gast System mitschneiden
- Wine
  - Falls der Treiber mit Wine lauffähig ist
- Windows Nativ installieren
  - Direkt mitschneiden
  - Kein Wireshark

# Projekt – USB Treiber



WebMail Notifier  
(Dream Cheeky)

## Probleme

- Nur Windows Treiber
- Keine Schnittstellen Beschreibung

# Links

- Official USB developer documentation  
<http://www.usb.org/developers/docs/>
- LibUSB Homepage  
<http://libusb.org/>
- WebMail Notifier Linux driver (Dream Cheeky)  
<https://github.com/daniel-git/usblamp>

# USB und libUSB

**Daniel Käfer**

Advanced Computer Science

Hochschule Furtwangen

[d.kaefer@hs-furtwangen.de](mailto:d.kaefer@hs-furtwangen.de)