Understand USB (in Linux)

Krzysztof Opasiak

Samsung R&D Institute Poland
Agenda

What USB is about?

Plug and Play

How BadUSB works?

May I have my own USB device?

Q & A
What USB is about?
What USB is about?

It is about providing services!

- Storage
- Printing
- Ethernet
- Camera
- Any other
How we connect them?
Logical vs physical topology

Physical

Logical
What is USB device?

- Piece of hardware for USB communication
- USB protocol implementation
- Some useful protocol implementation
- Piece of hardware/software for providing desired functionality
Endpoints...

- Device may have up to 31 endpoints (including ep0)
- Each of them gets a unique Endpoint address
- Endpoint 0 may transfer data in both directions
- All other endpoints may transfer data in one direction:
  - IN Transfer data from device to host
  - OUT Transfer data from host to device
Endpoint types

• **Control**
  • Bi-directional endpoint
  • Used for enumeration
  • Can be used for application

• **Interrupt**
  • Transfers a small amount of low-latency data
  • Reserves bandwidth on the bus
  • Used for time-sensitive data (HID)
Endpoint types

- **Bulk**
  - Used for large data transfers
  - Used for large, time-insensitive data (Network packets, Mass Storage, etc).
  - Does not reserve bandwidth on bus, uses whatever time is left over

- **Isochronous**
  - Transfers a large amount of time-sensitive data
  - Delivery is not guaranteed (no ACKs are sent)
  - Used for Audio and Video streams
  - Late data is as good as no data
  - Better to drop a frame than to delay and force a re-transmission
USB device
USB bus - low level

- USB is a Host-controlled bus
- Nothing on the bus happens without the host first initiating it.
- Devices cannot initiate any communication.
- The USB is a Polled Bus.
- The Host polls each device, requesting data or sending data.
Plug and Play

Embedded Linux Conference
Step by step

- Plug in device
- Detect Connection
- Set address
- Get device info
- Choose configuration
- Choose drivers for interfaces
- Use it ;}
Set address

- On plug-in device use default address 0x00
- Only one device is enumerated at once
- Hosts assigns unique address for new device
Get device info

- Each USB world entity is described by a data structure called descriptor.
- Descriptors have different types, sizes and content.
- But they all have a common header.

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of the Descriptor in Bytes</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>Device Descriptor (0x01)</td>
</tr>
<tr>
<td>&lt;data&gt;</td>
<td>bLength - 2</td>
<td>NA</td>
<td>Payload</td>
</tr>
</tbody>
</table>

Field bLength is 1 byte, and the Value is a Number indicating the Size of the Descriptor in Bytes. The bDescriptorType field is 1 byte, with a Constant Value of 0x01, indicating a Device Descriptor. The <data> field is bLength - 2 bytes, with a Value of NA, and represents the Payload data.
USB descriptors

USB Device Descriptor
- idVendor
- idProduct
- bDeviceClass
- bDeviceSubClass
- bDeviceProtocol
- bManufacturer
- bProduct
- bSerial
- bNumConfigurations

USB Configuration Descriptor
- bMax Power
- iConfiguration
- bNumInterfaces

USB Interface Descriptor
- bAlternateSetting
- bInterfaceClass
- bInterfaceSubClass
- bInterfaceProtocol
- bInterface
- bNumEndpoints

USB Endpoint Descriptor
- bEndpointAddress
- bmAttributes
- wMaxPacketSize
- bInterval
# USB classes

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Device</td>
<td>Use class information in the Interface Descriptors</td>
</tr>
<tr>
<td>01h</td>
<td>Interface</td>
<td>Audio</td>
</tr>
<tr>
<td>02h</td>
<td>Both</td>
<td>Communications and CDC Control</td>
</tr>
<tr>
<td>03h</td>
<td>Interface</td>
<td>HID (Human Interface Device)</td>
</tr>
<tr>
<td>05h</td>
<td>Interface</td>
<td>Physical</td>
</tr>
<tr>
<td>06h</td>
<td>Interface</td>
<td>Image</td>
</tr>
<tr>
<td>07h</td>
<td>Interface</td>
<td>Printer</td>
</tr>
<tr>
<td>08h</td>
<td>Interface</td>
<td>Mass Storage</td>
</tr>
<tr>
<td>09h</td>
<td>Device</td>
<td>Hub</td>
</tr>
<tr>
<td>0Ah</td>
<td>Interface</td>
<td>CDC-Data</td>
</tr>
<tr>
<td>0Bh</td>
<td>Interface</td>
<td>Smart Card</td>
</tr>
<tr>
<td>0Dh</td>
<td>Interface</td>
<td>Content Security</td>
</tr>
<tr>
<td>0Eh</td>
<td>Interface</td>
<td>Video</td>
</tr>
<tr>
<td>0Fh</td>
<td>Interface</td>
<td>Personal Healthcare</td>
</tr>
<tr>
<td>10h</td>
<td>Interface</td>
<td>Audio/Video Devices</td>
</tr>
<tr>
<td>11h</td>
<td>Device</td>
<td>Billboard Device Class</td>
</tr>
<tr>
<td>DCh</td>
<td>Both</td>
<td>Diagnostic Device</td>
</tr>
<tr>
<td>E0h</td>
<td>Interface</td>
<td>Wireless Controller</td>
</tr>
<tr>
<td>EFh</td>
<td>Both</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>FEh</td>
<td>Interface</td>
<td>Application Specific</td>
</tr>
<tr>
<td>FFh</td>
<td>Both</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>
Device Info Summary

- Host gets info about new devices from suitable USB descriptors
- Most important data at this moment:
  - idVendor
  - idProduct
  - bcdDevice
  - bDeviceClass
  - bDeviceSubClass
  - bDeviceProtocol
  - bMaxPower
  - bInterfaceClass
  - bInterfaceSubClass
  - bInterfaceProtocol
Set Configuration

• Which configuration is the most suitable?
  • We have enough power for it (bMaxPower?)
  • It has at least one interface
  • If device has only one config just use it
  • Choose the one which first interface is not Vendor specific

• All interfaces of choosen configuration becomes enabled so let's use them
What USB driver really is?

- Piece of kernel code
- Usually provides something to userspace (network interface, tty, etc.)
- Implementation of some communication protocol
How to choose a suitable driver?

- `struct usb_driver`
- **When device needs special handling:**
  - Using VID and PID and interface id
  - Driver probe()s for each interface in device that match VID and PID
- **When driver implements some well defined, standardized protocol**
  - Using bInterfaceClass, bInterfaceSubClass etc.
  - Driver probe() for each interface which has suitable identity
  - No matter what is the VID and PID
  - Driver will not match if interface hasn't suitable class
What's next?

• We have the driver which provides something to userspace but what's next?
• It depends on interface type:
  • Network devices - Network manager should handle new interface setup
  • Pendrives, disks etc - automount service should mount new block device
  • Mouse, keyboard - X11 will start listening for input events
  • And many many other things are going to be handled AUTOMATICALLY
  • without any user action…
How BadUSB works?
USB security summary

- Between plug in and start using there is no user interaction
- Drivers are probed automatically
- Userspace starts using new device automatically
- Device introduce itself as it wants
- There is no relation between physical outfit and descriptors
My beautiful tablet
BadUSB attack scenario

- User connect hacked device
- Device looks like pendrive, tablet…
- But sends descriptor taken from some keyboard
- And implements HID protocol
- Kernel creates new input source
- and X11 just starts using them
How dangerous it is?

- I just downloaded image and changed the background but what else it can do?
- There is a version of this attack which spoofs DNS on host and redirects them to USB device
- Any command which doesn't require sudo can be executed
  - anything!
  - anything!
  - anything!
How to protect?

- Don't connect unknown devices found on a street
- Limit number of input source to X11
- Use device/interface authorization
  - usbguard
  - gnome solution
Device/interface authorization

- Each USB device has *authorized* attribute in sysfs directory
- Each HCD has *authorized_default* entry in sysfs
- If we set this to false each new device on this bus will be unauthorized by default
- Drivers will not be able to bind to it
- This gives us time to use *lsusb* to check it
My tablet (once again)
May I have my own USB device?
Yes, you can!

<table>
<thead>
<tr>
<th>Need</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suitable hardware</td>
<td>Get some board with UDC controller (BBB, Odroid etc.)</td>
</tr>
<tr>
<td>Implementation of USB protocol</td>
<td>Use one from Linux kernel!</td>
</tr>
<tr>
<td>Implementation of some useful</td>
<td>A lot of protocols are available out of the box in Linux kernel!</td>
</tr>
<tr>
<td>protocol</td>
<td></td>
</tr>
<tr>
<td>Desired functionality provider</td>
<td>Let's use our system infrastructure!</td>
</tr>
</tbody>
</table>
Terminology

USB device = USB gadget + UDC

UDC driver  Driver for USB Device Controller

USB function (type)  driver which implements some useful protocol (HID, Mass storage)

USB gadget  Glue layer for functions.
  • Handle enumeration
  • Respond to most general requests
Device architecture overview

- USB bus
  - UDC driver
  - USB core
  - USB gadget driver
  - USB function
  - USB function
  - network interface block device character device etc.
  - network interface block device character device etc.

User
Prerequisites - menuconfig
Available functions

- Ethernet
  - ECM
  - EEM
  - NCM
  - Subset
  - RNDIS
- Serial
  - ACM
  - Serial
  - OBEX
- Mass Storage
- HID
- UVC
- UAC
- Printer
- Phonet
- Loopback and SourceSink
Base composition

- Fill the identity of gadget
  - Vendor ID
  - Product ID
  - Device Class details
  - Strings (manufacturer, product and serial)
- Decide what functions
- Decide how many configurations
- Decide what functions are available in each configuration
But how to do this?

• Use bare kernel ConfigFS interface
  Documentation/ABI/testing/configfs-usb-gadget*

• Use libusbgx to create a program
  https://github.com/libusbgx/libusbgx

• Use gt to create a simple script
  https://github.com/kopasiak/gt

• Use gt to load gadget scheme
What gadget schemes really are?

• Declarative gadget description
• Simple configuration file
• libconfig syntax
• Interpreted by libusbgx
• Can be easily loaded using gt load

```plaintext
attrs = {
    idVendor = 0x1D6B
    idProduct = 0xe1ce
}
strings = {
    lang = 0x409;
    manufacturer = "Linux Kernel"
    product = "Sample gadget"
    serialnumber = "ELC2016"
}
functions = {
    our_net = {
        instance = "net1"
        type = "ecm"
    }
}
configs = {
    id = 1
    name = "c"
    strings = {
        lang = 0x409
        configuration = "The only one"
    }
    functions = ("our_net")
}
```
Let's compose some device
Q & A
Thank you!

Krzysztof Opasiak
Samsung R&D Institute Poland

+48 605 125 174
k.opasiak@samsung.com
References

• Tame The USB gadgets Talkative Beast, Krzysztof Opasiak
• Make your own USB gadget, Andrzej Pietrasiewicz
• USB and the Real World, Alan Ott
• USB in a Nutshell
• USB specification
• BadUSB attack
• usbguard
• libubsgx
• gt