Understand USB (in Linux)

Krzysztof Opasiak

SAMSUNG

Samsung R&D Institute Poland

Embedded Linux Conference

Agenda

What USB is about?

Plug and Play

How BadUSB works?

May I have my own USB device?

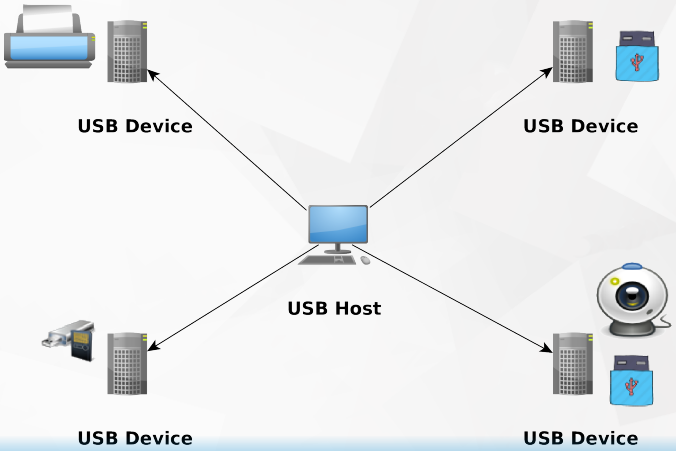
Q & A

What USB is about?

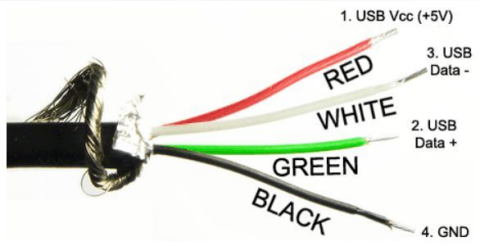
What USB is about?

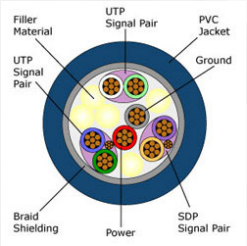
It is about providing services!

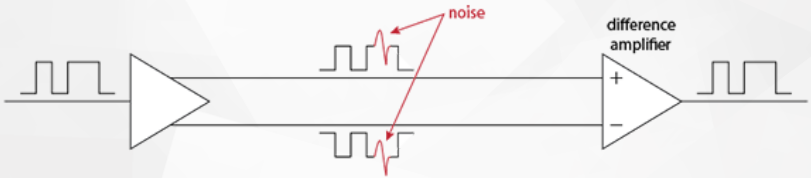
- **Storage**
- **Printing**
- **Ethernet**
- **Camera**
- **Any other**



How we connect them?

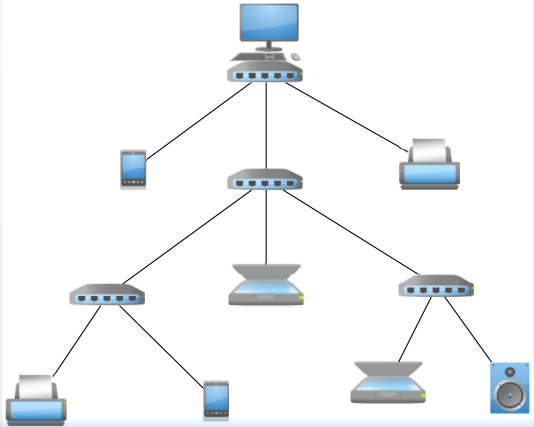




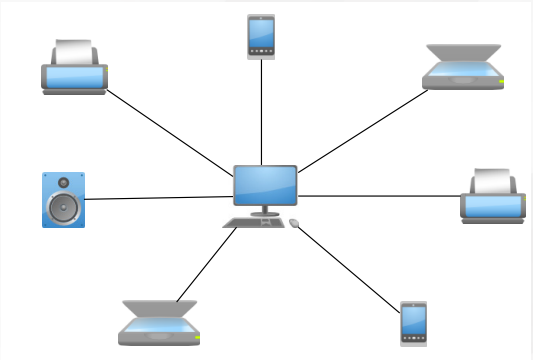


Logical vs physical topology

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

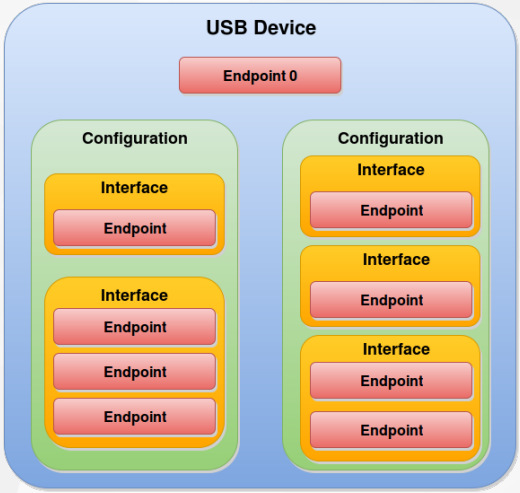
Endpoint types

- **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 Device
Endpoint 0
Configuration
Interface
Endpoint
Interface
Endpoint
Endpoint
Endpoint
Configuration
Interface
Endpoint
Interface
Endpoint
Interface
Endpoint
Endpoint

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

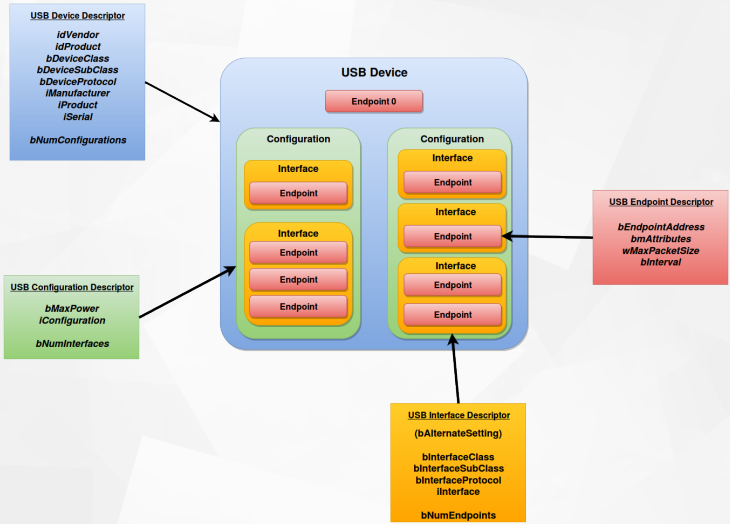
- On plug-in device use default address 0x00
- Only one device is enumerated at once
- Hosts assigns unique address for new device

Get device info

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

Field	Size	Value	Description
bLength	1	Number	Size of the Descriptor in Bytes
bDescriptorType	1	Constant	Device Descriptor (0x01)
<data>	bLength - 2	NA	Payload

USB descriptors



USB classes

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

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**

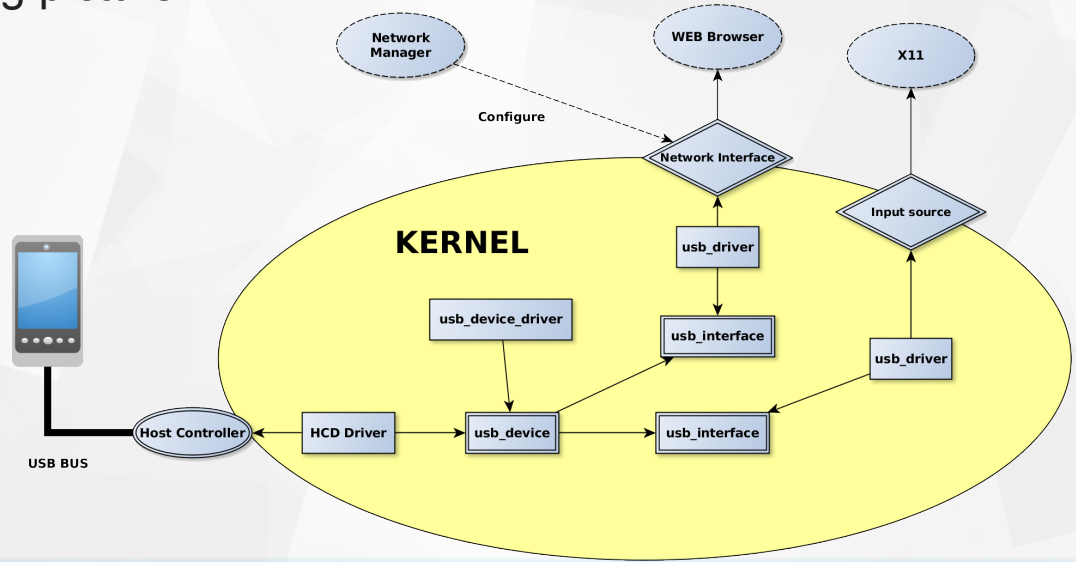
What USB driver really is?

- **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

Big picture


Network Manager
WEB Browser
X11
Configure
Network Interface
Input source
KERNEL
usb_driver
usb_device_driver
usb_interface
usb_driver
Host Controller
HCD Driver
usb_device
usb_interface
USB BUS

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!

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

Terminology

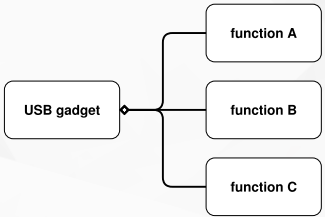
USB device = USB gadget + UDC

UDC driver Driver for USB Device Controller

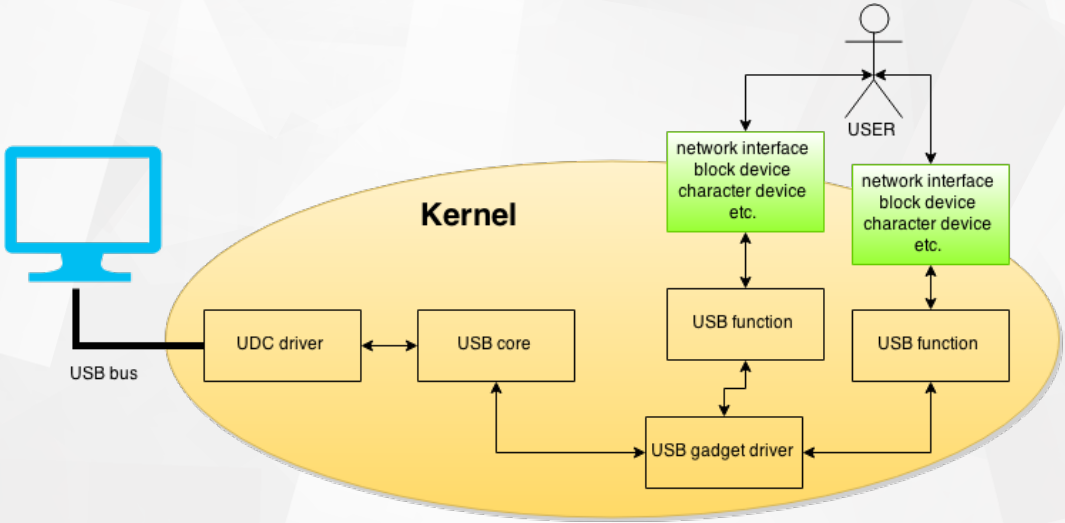
USB function (type) driver which implements some useful protocol (HID, Mass storage)

USB gadget Glue layer for functions.

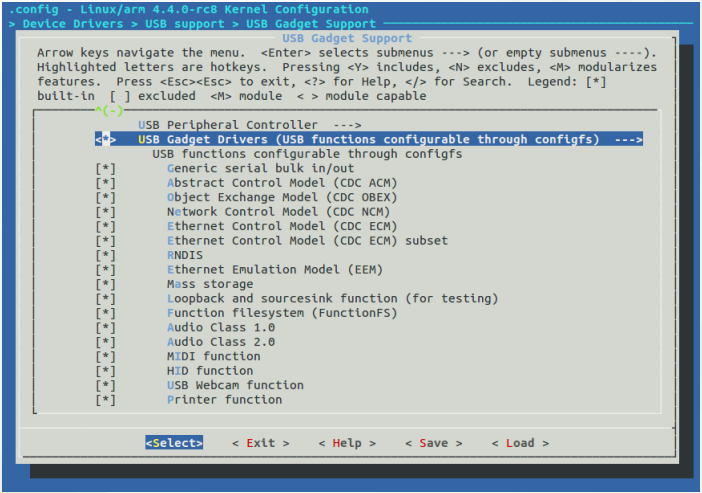
- **Handle enumeration**
- **Respond to most general requests**



Device architecture overview

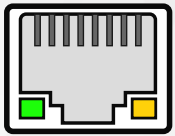


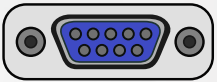
Prerequisites - menuconfig



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**

```
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

Embedded Linux Conference

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