USB INTERFACING AND DRIVERS

Team Members:
Koshik Samota  (1205087542)
Jubin Mehta    (1205367016)

Mentor:
Dr. Yann Hang Lee
WHAT IS USB?

You think USB is this →

BUT, actually USB is an entire Communication Protocol!!
WHY USB?

- Earlier, connecting external devices to a host computer was very complex.
- They used different ports and protocols.
- E.g.: Mice, Keyboards, Printers, etc.
- But, now USB has standardized all these connections. All the devices that used different ports and protocols now use a standard Interface – ‘USB’
WHY USB? (CONTD…)

<table>
<thead>
<tr>
<th>Connection / Port</th>
<th>Maximum Speed</th>
<th>Connector Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Parallel Port</td>
<td>0.115 MBytes/s</td>
<td>25</td>
</tr>
<tr>
<td>Enhanced Parallel Port</td>
<td>3.0 Mbytes/s</td>
<td>25</td>
</tr>
<tr>
<td>Extended Capabilities Port</td>
<td>3.0 Mbytes/s</td>
<td>25</td>
</tr>
<tr>
<td>Standard RS-232 COM Port</td>
<td>0.03 Mbytes/s</td>
<td>9</td>
</tr>
<tr>
<td>PS-2 Port</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>60 Mbytes/s</td>
<td>4</td>
</tr>
</tbody>
</table>

USB provide us with High Speed with Fewer Wires
## EVOLUTION

<table>
<thead>
<tr>
<th>Standard</th>
<th>Year</th>
<th>Speed</th>
<th>Use</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB 1.0</td>
<td>Jan 1996</td>
<td>1.5 Mbps (Low Speed)</td>
<td>USB Mice USB Keyboard</td>
<td>![USB Symbol]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB 1.1</td>
<td>July 1998</td>
<td>12 Mbps (Full Speed)</td>
<td>USB Storage</td>
<td>![USB Symbol]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB 2.0</td>
<td>April 2000</td>
<td>480 Mbps (High Speed)</td>
<td>USB Storage Devices</td>
<td>![USB Symbol]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB OTG</td>
<td>July 2003</td>
<td>480 Mbps (High Speed)</td>
<td>Intended to share data w/o Computer</td>
<td>![USB Symbol]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USB 3.0</td>
<td>Dec 2008</td>
<td>5 Gbps (Super Speed)</td>
<td>USB Storage Devices</td>
<td>![USB Symbol]</td>
</tr>
</tbody>
</table>
Low Speed and Full Speed USB Devices are Compatible with USB 2.0 (High Speed) Host Controller.

* as per USB 2.0 specifications
BUS TOPOLOGY

- USB physical interconnect is a tiered star topology.
- A hub is at the center of each star which can further connect to another hub or a device.
- Due to timing constraints allowed, the maximum number of tiers allowed is 7 and the maximum number of devices that can be connected is 128.
- A compound device is a combination of a root and device.

* as per USB 2.0 specifications
ADDRESSING IN USB:

ENDPOINT: An addressable unit in USB.
- There are 32 endpoints available.
- ENDPOINT can be either IN or OUT
  - IN: Host ← Device
  - OUT: Host → Device
- Each endpoint address has an associated data transfer type.
- Endpoint 0 is exclusively used as Default Control Pipe in Device Configuration.

PIPES: Logical Connection between 2 ENDPOINTS
- Pipes are of 2 types
  - Message Pipe:
    - Bi-directional data transfer
    - Used for small commands and responses (Control)
  - Stream Pipe:
    - Unidirectional data transfer
    - Used for actual data transfer (Bulk, Isochronous, Interrupt)

* as per USB 2.0 specifications
Control transfers
It is used for non-periodic control, status and configuration information

Bulk transfers
It is used for large chunks of time-insensitive data

Isochronous transfers
It is used for periodic time-critical data transfer

Interrupt transfers
Used to exchange small quantities of time-sensitive data

USB Data Transfer types

* as per USB 2.0 specifications
DATA STRUCTURES USED

```c
struct usb_device {
    /* ... */
    enum usb_device_state state;
    enum usb_device_speed speed;
    /* ... */
    struct usb_device *parent;
    /* ... */
    struct usb_device_descriptor descriptor;
    struct usb_host_config *config;
    /* ... */
    int maxchild;
    struct usb_device *children[USB_MAXCHILDREN]
    /* ... */
};
```

Description:

(Important parameters only)

- Status: Configured or not
- Speed: High/Full/Low
- Pointer to parent hub
- Own Device Descriptor
- Pointers to the child devices (in case of HUB)

* as per USB 2.0 specifications
DATA STRUCTURES USED

struct urb
{
    struct kref kref;
    /* ... */
    struct usb_device *dev;
    unsigned int pipe;
    int status;
    void *transfer_buffer;
    dma_addr_t transfer_dma;
    int transfer_buffer_length;
    /* ... */
    int interval;
    /* ... */
    void *context;
    usb_complete_t complete;
    /* ... */
};

Description:

(Important parameters only)

• No. of Requests present
• Pointer to usb_device
• Pipe information
• Pointer to associated Data Buffer
• DMA address for transfer
• Polling interval for interrupt or isochronous transfers
• Pointer to the specific driver context
• Pointer Callback Function Handler

* as per USB 2.0 specifications
DATA STRUCTURES USED

LIFE CYCLE OF URB

CREATE
- Allocates memory
- Initializes kobject
- Initializes spinlock
- `usb_alloc_urb()`

POPULATE
- Initializes all the members of the `struct urb`
- `usb_fill_[control|int|bulk]_urb()`

SUBMIT
- Submits urb for data transfer
- It is asynchronous
- `usb_submit_urb()`
- Callback function is executed at last to verify the submission

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

USB Host Controller:

- It is the hardware and software that allows USB devices to be attached to a host.
- It manages the data flow and bus access to the USB Device.

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

USB HOST CONTROLLER DRIVER:

- It provides an abstraction interface between the USB Host Controller and USB Core.
- This interface supports different Host Controllers without requiring specific knowledge of a Host Controller implementation.
- Transfers the data in USB FORMAT, i.e., URBs.

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

* as per USB 2.0 specifications
**DRIVER ARCHITECTURE**

**USB Core:**

- The core also provides a level of indirection that renders client drivers independent of host controllers.
- It basically has routines and structures available to both the host controller drivers and client drivers.

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

* as per USB 2.0 specifications
**DRIVER ARCHITECTURE**

**USB CLIENT DRIVER:**

- It is the software that executes on the host that supports the specific USB device.
- It is either supplied by the host operating system or fetched from the USB device itself.

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

khubd:

- It is used as detecting changes in the port status and the feature of plug n play is time-consuming for the hub driver.
- It is asleep by default.
- Hub driver wakes it up on a change in port status.

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

* as per USB 2.0 specifications
DRIVER ARCHITECTURE

USB FILE SYSTEM:

• It helps the user to access the USB device from the user space.

* as per USB 2.0 specifications
USB : PLUG AND PLAY DEVICE

• This feature allows a new USB hardware device to be connected to a system without the intervention of a USER.

• Device Drivers are found:
  • Pre – Installed on the System.
  • Downloaded via the INTERNET.
  • Fetched from the Device.

* as per USB 2.0 specifications
DETECTION OF USB DEVICE: ENUMERATION

Step 0
- The life of a **Hot plugged** USB Device starts with **ENUMERATION**.
- The host learns about the capabilities of the device and configures the device.

Step 1
- It starts, when the Root Hub reports a **change in the Port’s status** due to Device Attachment.
- It changes its status to USB_PORT_STAT_C_CONNECTION and awakens **khubd**, a helper thread.

Step 2
- Khubd is responsible to trace the identity of the port where the device is connected
- It randomly assigns an address between **1 and 127** and assigns to the bulk endpoint using a control URB attached to **endpoint 0**

Step 3
- Khubd then, gets the **device descriptor** from the control URB attached at Endpoint 0.
- It then requests for the **device’s configuration** descriptors and selects a suitable one.

Step 4
- Khubd then requests the **Core** to attach a matching **Client Driver** for the Device.

Step 5
- When the enumeration is complete, the khubd invokes the **probe()** method described in the **Client Driver** for normal execution.

* as per USB 2.0 specifications
REFERENCES:

[1] Essential Linux Device Drivers - Sreekrishnan Venkateswaran
[6] Embedded USB Host Stack - Steffen Weiss, Guenter Hildebrandt
[8] www.usb.org
thank you!