USB core callbacks

What callbacks will usbcore do?

Usbcore will call into a driver through callbacks defined in the driver structure and through the completion handler of URBs a driver submits. Only the former are in the scope of this document. These two kinds of callbacks are completely independent of each other. Information on the completion callback can be found in :ref:`usb-urb`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]callbacks.rst`, line 7); *backlink*
>
> Unknown interpreted text role "ref".

The callbacks defined in the driver structure are:

1. Hotplugging callbacks:

   - @probe:
     Called to see if the driver is willing to manage a particular interface on a device.
   - @disconnect:
     Called when the interface is no longer accessible, usually because its device has been (or is being) disconnected or the driver module is being unloaded.

2. Odd backdoor through usbfs:

   - @ioctl:
     Used for drivers that want to talk to userspace through the "usbfs" filesystem. This lets devices provide ways to expose information to user space regardless of where they do (or don't) show up otherwise in the filesystem.

3. Power management (PM) callbacks:

   - @suspend:
     Called when the device is going to be suspended.
   - @resume:
     Called when the device is being resumed.
   - @reset_resume:
     Called when the suspended device has been reset instead of being resumed.

4. Device level operations:

   - @pre_reset:
     Called when the device is about to be reset.
   - @post_reset:
     Called after the device has been reset

The ioctl interface (2) should be used only if you have a very good reason. Sysfs is preferred these days. The PM callbacks are covered separately in :ref:`usb-power-management`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\usb\[linux-master][Documentation][driver-api][usb]callbacks.rst`, line 54); *backlink*
>
> Unknown interpreted text role "ref".

Calling conventions

All callbacks are mutually exclusive. There's no need for locking against other USB callbacks. All callbacks are called from a task context. You may sleep. However, it is important that all sleeps have a small fixed upper limit in time. In particular you must not call out to user space and await results.

Hotplugging callbacks

These callbacks are intended to associate and disassociate a driver with an interface. A driver's bond to an interface is exclusive.

The probe() callback

```
int (*probe) (struct usb_interface *intf,
              const struct usb_device_id *id);
```

Accept or decline an interface. If you accept the device return 0, otherwise -ENODEV or -ENXIO. Other error codes should be used only if a genuine error occurred during initialisation which prevented a driver from accepting a device that would else have been accepted. You are strongly encouraged to use usbcore's facility, usb_set_intfdata(), to associate a data structure with an interface, so that you know which internal state and identity you associate with a particular interface. The device will not be suspended and you may do IO to the interface you are called for and endpoint 0 of the device. Device initialisation that doesn't take too long is a good idea here.

The disconnect() callback

```
void (*disconnect) (struct usb_interface *intf);
```

This callback is a signal to break any connection with an interface. You are not allowed any IO to a device after returning from this callback. You also may not do any other operation that may interfere with another driver bound the interface, eg. a power management operation. If you are called due to a physical disconnection, all your URBs will be killed by usbcore. Note that in this case disconnect will be called some time after the physical disconnection. Thus your driver must be prepared to deal with failing IO even prior to the callback.

Device level callbacks

pre_reset

```
int (*pre_reset)(struct usb_interface *intf);
```

A driver or user space is triggering a reset on the device which contains the interface passed as an argument. Cease IO, wait for all outstanding URBs to complete, and save any device state you need to restore. No more URBs may be submitted until the post_reset method is called.

If you need to allocate memory here, use GFP_NOIO or GFP_ATOMIC, if you are in atomic context.

post_reset

```
int (*post_reset)(struct usb_interface *intf);
```

The reset has completed. Restore any saved device state and begin using the device again.

If you need to allocate memory here, use GFP_NOIO or GFP_ATOMIC, if you are in atomic context.

Call sequences

No callbacks other than probe will be invoked for an interface that isn't bound to your driver.

Probe will never be called for an interface bound to a driver. Hence following a successful probe, disconnect will be called before there is another probe for the same interface.

Once your driver is bound to an interface, disconnect can be called at any time except in between pre_reset and post_reset. pre_reset is always followed by post_reset, even if the reset failed or the device has been unplugged.

suspend is always followed by one of: resume, reset_resume, or disconnect.