USB Theory

Basic Knowledge Required Before Writing a USB Driver

USB Enumeration

- USB enumeration happens when a device is first plugged in.
- Enumeration is a complex process and can vary between operating systems.
- By the time enumeration has completed, the host knows the device's vendor ID, product ID, connection speed, endpoint addresses and endpoint types.
- Enumeration is handled by the USB core. No driver interaction required.

USB Device Necessities

- Endpoint 0. Endpoint 0 is used to read and write configuration information to the device. Also used for vendor commands.
- Vendor ID and product ID. Each USB device must have a 16-bit vendor ID and 16-bit product ID. These IDs are used by the USB core to determine which driver to use for the device.

USB Data Transfer Types

- Bulk - Used to transfer periodic data. Most USB devices use bulk endpoints (misc. data transfers).
- Interrupt - Used to transfer small amounts of sporadic data. Human interface devices use interrupt endpoints (mouse, keyboard).
- Isochronous - Used for data streaming (audio and video devices).
- Vendor commands. Simple commands used to configure the USB device.

USB Request Block (URB)

- The URB is a message passing device used by the USB core. All USB device transactions are handled through URBs therefore the USB driver must use URBs for device interaction. URBs are queued by the USB core and sent to the appropriate device.

Linux USB Drivers

Major Functions and Steps Required
to Make a USB Driver Work

Major Structs

- Device context structure. User defined structure. This is a big structure and contains all the specific information about the USB device such as device pointer, interface pointer, endpoint information, URB pointers, locks, and any other information the driver might want to keep track of.
- struct usb_device_id. Contains a list of devices that work with this driver. The information in this table is prepared for the USB core with the MODULE_DEVICE_TABLE macro.
- struct file_operations. Contains all the valid file operations that can be performed on the USB device.
- struct usb_driver. Contains the name of the device and function pointers to the various USB functions (probe, disconnect, suspend, resume, etc.).
- struct usb_class_driver. Used to register the USB device with devfs and the driver core.

Major Functions and Function Summaries

- **init_module.** Called when the USB driver is inserted into the kernel.

    1. Registers the driver with the USB core (usb_register).

- **cleanup_module.** Called when the USB driver is removed from the kernel.

    1. Deregisters the driver with the USB core (usb_deregister).

Major Functions (Continued)

- **probe.** Called when the USB device is connected to the host and after enumeration has occurred.
    1. Create and initialize the device context structure.
    2. Create sysfs files for the various device components (device_create_file). These files appear in the sys/class/usb/(usb device)/device/ directory.
    3. Increment through struct usb_interface->cur_altsetting-> desc.bNumEndpoints to find the various endpoint addresses and types. Store this information in the device context structure. A pointer to struct usb_interface is passed by the USB core to the probe function.
    4. Initialize interrupts (usb_rcvintpipe).
    5. Create interrupt endpoint buffer.
    6. Create interrupt endpoint URB (usb_alloc_urb).
    7. Fill interrupt endpoint URB (usb_fill_int_urb).
    8. Submit interrupt URB to USB core (usb_submit_urb).
    9. Create bulk endpoint buffers.
    10. Register device (usb_register_dev).

Major Functions (Continued)

- **disconnect.** Called when the device is unplugged from the host.
  2. Release interface resources (usb_put_dev, usb_free_urb, usb_set_intfdata).
  3. Return minor number to driver core (usb_deregister_dev).
  4. Release all URB resources (usb_kill_urb).
  5. Remove files from sysfs (device_remove_file).
  6. Decrement device reference count (kref_put).
- **suspend**
  1. Stop the interrupt URB (usb_kill_urb).
- **resume**
  1. Restart the interrupt URB (usb_submit_urb).

Major Functions (Continued)

- **open.** Called when /dev/osrfx2_0 is opened.
    1. Reset bulk out pipe (usb_clear_halt).
    2. Reset bulk in pipe (usb_clear_halt).
    3. Increment device reference count (kref_get).
    4. Save pointer to device instance for future reference.
- **close.** Called when /dev/osrfx2_0 is closed.
    1. Clear bulk read and bulk write available status.
    2. Decrement device reference count (kref_put).

Major Functions (Continued)

- **read.** Called when /dev/osrfx2_0 is read from.
    1. Initialize pipe (usb_rcvbulkpipe).
    2. Read bulk in data (usb_bulk_msg).
    3. Copy data to user space (copy_to_user).
- **write.** Called when data is written to /dev/osrfx2_0.
    1. Create URB (usb_alloc_urb).
    2. Create URB buffer (usb_alloc_coherent).
    3. Copy data to write buffer (copy_from_user).
    4. Initialize the URB (usb_sndbulkpipe, usb_fill_bulk_urb).
    5. Send the data to the device (usb_submit_urb).
    6. Release the URB (usb_free_urb).
- **write_callback**
    1. Check for device errors that may have occurred during the write.
    2. Release the write buffer (usb_free_coherent).

Major Functions (Continued)

- **interrupt_handler.** Called when interrupt received from device.
    1. Get interrupt data.
    2. Restart interrupt URB (usb_submit_urb).
- **Vendor commands.** Vendor commands are sent using the usb_control_msg function.
- **Create device attributes in the sysfs.** When the attribute files are read and written in the sysfs, corresponding get and set commands are called to control the 7 segment display, bargraph and switches (DEVICE_ATTR).

Misc. Linux USB Information

- USB_CONFIG. In order to do USB development in Linux, USB_CONFIG must be set to ‘y’ in the .config file before the kernel is compiled. Currently there is a bug in the latest Linux kernels where this value is set to ‘m’ after make menuconfig is run. Patches are available to fix this or USB_CONFIG can be manually set to ‘y’ by editing the .config file.
- Gadgets. A selection under the USB menu in menuconfig is “gadgets”. By default it is not selected. By selecting gadgets, device side USB support will be enabled in the kernel. Used in embedded Linux.
- /dev/(usb device). Used to read and write from bulk endpoints.
- /sys/class/usb/(usb device)/device/. Reading and writing to these files can call individual vendor commands or provide various information about the device.

References

- /usr/src/(kernel version)/drivers/usb/usb-skeleton.c – Basic USB driver.
- /usr/src/(kernel version)/drivers/usb/atm/usbatm.c – Isochronous endpoints.
- http://www.linuxjournal.com/article/4786 - How to write a USB driver.
- http://www.linux-usb.org/ - Linux USB driver support.
- http://lwn.net/images/pdf/LDD3/ch13.pdf - Linux device drivers book.
- http://www.mjmwired.net/kernel/Documentation/usb/URB.txt - URB documentation.