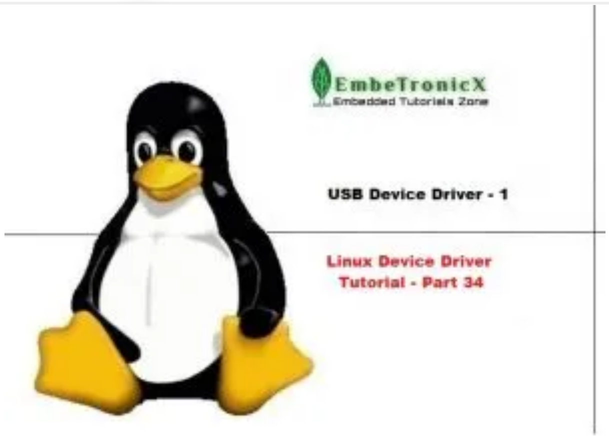
EmbeTronicX
Embedded Tutorials Zone

Sidebar

Home → Tutorials → Linux → Device Drivers → **Linux Device Driver Tutorial Part 33 – USB Device Driver Basics 1**

Device Drivers



Linux Device Driver Tutorial Part 33 – USB Device Driver Basics 1

This is the Series on Linux Device Driver. The aim of this series is to provide easy and practical examples that anyone can understand. This is the Linux Device Driver Tutorial Part 33 – USB Device Driver Basics in the Linux Device Driver.

Post Contents [hide]

Prerequisites

You should have the basic knowledge of the below topics before writing the USB driver.

- USB protocol specification

USB Device Driver Basics

Introduction

At the early stages of the Linux kernel, the USB subsystem was supporting only mice and keyboards. Then later on it continues to grow. Right now it is supporting plenty of USB devices. Before writing the USB device driver, we should discuss how the USB devices are connected to the Linux system and what is happening while plugging into the system. For that, you need to understand the USB Subsystem in Linux.

USB subsystem consists of the USB core, the USB Host Controller, and the USB host controller's driver.

Just see the below image to get some idea.

Here we will discuss one by one.

USB Device

This is just a normal USB device like Pendrive, Mice, Keyboards, USB to Serial converter, USB MP3 Player, etc.

USB Host Controller

The USB host controller is used to control the communications between the host system and USB devices. This USB host controller contains both hardware and software parts.

Hardware:

This hardware part is used for the below operations.

- Detecting the connected and disconnected USB devices.
- Providing power to the connected USB devices.
- and so on.

Software:

The software part is also called as a USB host controller driver. This will be used to loading the correct USB device drivers, and managing the data transfer between the driver and USB host controller.

So this USB host controller is doing the initial transaction once the USB device is detected in the system. Here Initial transaction means, get the devices vendor id, get the devices product id, device type, etc. Once that is done, then the USB host controller driver will assign the appropriate USB device driver to the device.

There are multiple USB host controller interfaces are available.

- Open Host Controller Interface (OHCI) – USB 1.X. eg: USB 1.1
- Universal Host Controller Interface (UHCI) – USB 1.X.
- Enhanced Host Controller Interface (EHCI) – USB 2.0.
- Extended Host Controller Interface (XHCI) – USB 3.0.

I am not going into deep about the above variants.

USB core

USB core is a codebase consisting of routines and structures available to HCDs (Host Controller Driver) and USB drivers.

USB Driver

This is the USB driver which we are going to write for the USB devices.

USB descriptor

Till now, we have some ideas about the USB and its subsystem in Linux. Now Let's discuss the USB device. The USB device contains a number of descriptors that help to define what the device is capable of. We will discuss about the below descriptors.

- Device descriptor
- Configuration descriptor
- Interface descriptor
- Endpoint descriptor

Device descriptor

USB devices can only have one device descriptor. The device descriptor includes information such as what USB revision the device complies with, the Product and Vendor IDs used to load the appropriate drivers, and the number of possible configuration descriptors the device can have.

Configuration descriptor

The configuration descriptor specifies values such as the amount of power this particular configuration uses if the device is self or bus-powered and the number of interfaces it has. When a device is enumerated, the host reads the device descriptors and can make a decision of which configuration to enable. A device can have more than one configuration, though it can enable only one configuration at a time.

Interface descriptor

A device can have one or more interfaces. Each interface descriptor can have a number of endpoints and represents a functional unit belonging to a particular class. For example you could have a multi-function fax/scanner/printer device. Interface descriptor one could describe the endpoints of the fax function, Interface descriptor two the scanner function, and Interface descriptor three the printer function. Unlike the configuration descriptor, there is no limitation as to having only one interface enabled at a time. A device could have 1 or many interface descriptors enabled at once.

Endpoint descriptor

Each endpoint descriptor is used to specify the type of transfer, direction, polling interval, and maximum packet size for each endpoint. In other words, each endpoint is a source or sink of data.

Please refer to the below image to get a better understanding.

Data Flow Types

There are four different ways to transfer data on a USB bus. Each has its own purposes and characteristics. Each one is built up using one or more transaction types. These data transfer types are set in the endpoint descriptor.

- Control Transfers
- Interrupt Transfers
- Bulk Transfers
- Isochronous Transfers

Control Transfers

Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed using control transfers.

This is a bi-directional transfer which uses both an IN and an OUT endpoint.

Interrupt Transfers

Interrupt transfers have nothing to do with interrupts. The name is chosen because they are used for the sort of purpose where an interrupt would have been used in earlier connection types.

Interrupt transfers are regularly scheduled IN or OUT transactions, although the IN direction is the more common usage.

Bulk Transfers

Bulk transfers are designed to transfer large amounts of data with error-free delivery, but with no guarantee of bandwidth. The host will schedule bulk transfers after the other transfer types have been allocated.

If an OUT endpoint is defined as using Bulk transfers, then the host will transfer data to it using OUT transactions.

If an IN endpoint is defined as using Bulk transfers, then the host will transfer data from it using IN transactions.

Isochronous Transfers

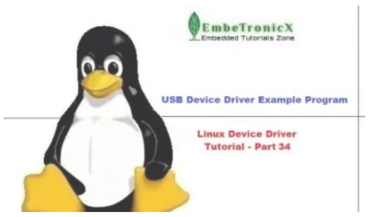
Isochronous transfers have a guaranteed bandwidth, but error-free delivery is not guaranteed.

The main purpose of isochronous transfers is applications such as audio data transfer, where it is important to maintain the data flow, but not so important if some data gets missed or corrupted.

An isochronous transfer uses either an IN transaction or an OUT transaction depending on the type of endpoint.

Oops, Hey Buddy, Wakeup. Such a long theory. Sorry about that. In our next tutorial, we will discuss about the programming of USB device drivers in Linux.

Related



Linux Device Driver Tutorial Part 34 – USB Device Driver Example - 2
In "Device Drivers"

Linux Device Driver Part 1 : Introduction
In "Device Drivers"

Linux Device Driver Tutorial Part 32 – Misc Device Driver
In "Device Drivers"

Subscribe

Connect with | Login

Be the First to Comment!

B I U S {} [+]

0 COMMENTS

report this ad