Generic USB Driver

Operating Systems Janki Trivedi 121018 | Shoghi Bagul 121051

Introduction

The Linux USB system has shown a growth for the support of device type from only supporting two types in the 2.2.7 kernel to supporting of over 20 different type of devices in 2.4 kernel. At present Linux system supports almost all kind of USB devices which include standard keyboards, mouse, modems, printers, speakers, digital cameras, Ethernet devices and MP3 players. There are vendor-specific devices that do not support the USB system in Linux because each vendor specifies a protocol to interact with its respective USB device which usually is a custom driver made by the vendor to be installed into the system before using the device.

What is a generic USB Driver?

Creating a generic driver is more beneficial than to install a separate driver for each device to be interfaced with the system. A generic USB driver will allow devices to interface with the Linux USB subsystem without the need to install any custom protocol from the vendor. The driver would support most of devices of a similar kind, for this case we consider USB flash storage devices. For example any USB mass storage from any vendor and manufacturer would at least get itself registered into the Linux USB Subsystem.

How does the generic USB Driver work?

The generic USB device driver at first needs to register itself with the Linux USB subsystem which provides information about the devices that are supported by the driver and the function needs to be called when a device interfaces such as inserted or removed from the system using the USB driver. All the info is stored into a skeleton structure as follows.

The variable name is a string that describes the USB driver. This name is used for informational purposes when printed in system log. The “probe” and “disconnect” function pointers are called when a device matches with the value provided in the id_table variable which is either seen .i.e. inserted or the device is removed from the system.

The USB driver is then registered with a call to the usb_register() function which is present in the USB driver's init() function.

Register and Deregister the USB Driver

Whenever a device is loaded into the system to enable the hot plug system automatically, a MODULE_DEVICE_TABLE is created. The following struct usb_device_id genericUSB_table[] consists of the vendor id and the product id

```
static struct usb_device_id genericUSB_table[] =
{
{ USB_DEVICE(0x058F, 0x6387) },
        {} /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, genericUSB_table);
```

When the USB driver is loaded into the system, the driver registers itself into the Linux USB subsystem using the usb_register() function.

```
  static int __init genericUSB_init(void)
{
    return usb_register(&genericUSB_driver);
}
```

When the USB driver is unloaded from the system, the driver unregisters itself from the Linux USB subsystem using the usb_deregister() function.

```
static void __exit genericUSB_exit(void)
{
        usb_deregister(&genericUSB_driver);
}
```

When a device is plugged into the USB bus that matches the device ID pattern that your driver registered with the USB core, the probe function is called. The usb_device structure, interface number and the interface ID are passed to the function:

genericUSB_probe() function

```
static int genericUSB_probe(struct usb_interface *interface, const struct usb_device_id *id)
```

The driver now needs to verify that this device is actually one that it can accept. If not, or if any error occurs during initialization, a NULL value is returned from the probe function. A pointer to a private data structure containing the driver's state for this device is returned. That pointer is stored in the usb_device structure, and all call-backs to the driver pass that pointer.

genericUSB_disconnect() function

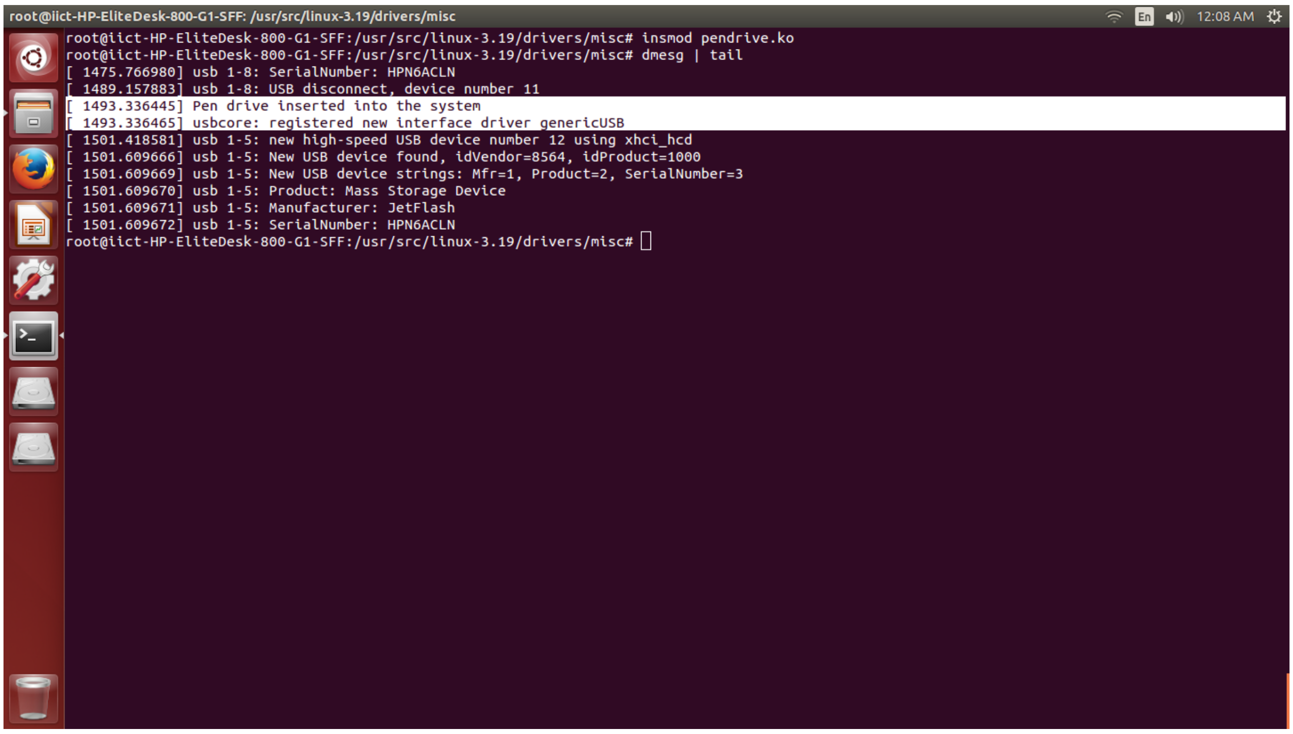
```
static void genericUSB_disconnect(struct usb_interface *interface)
```

If a program currently has an open handle to the device, we only null the usb_device structure in our local structure, as it has now gone away. For every read, write, release and other functions that expect a device to be present, the driver first checks to see if this usb_device structure is still present. If not, it releases that the device has disappeared, and a -ENODEV error is returned to the user-space program.

When the release function is eventually called, it determines if there is no usb_device structure and if not, it does the cleanup that the skel_disconnect function normally does if there are no open files on the device.

Implementation

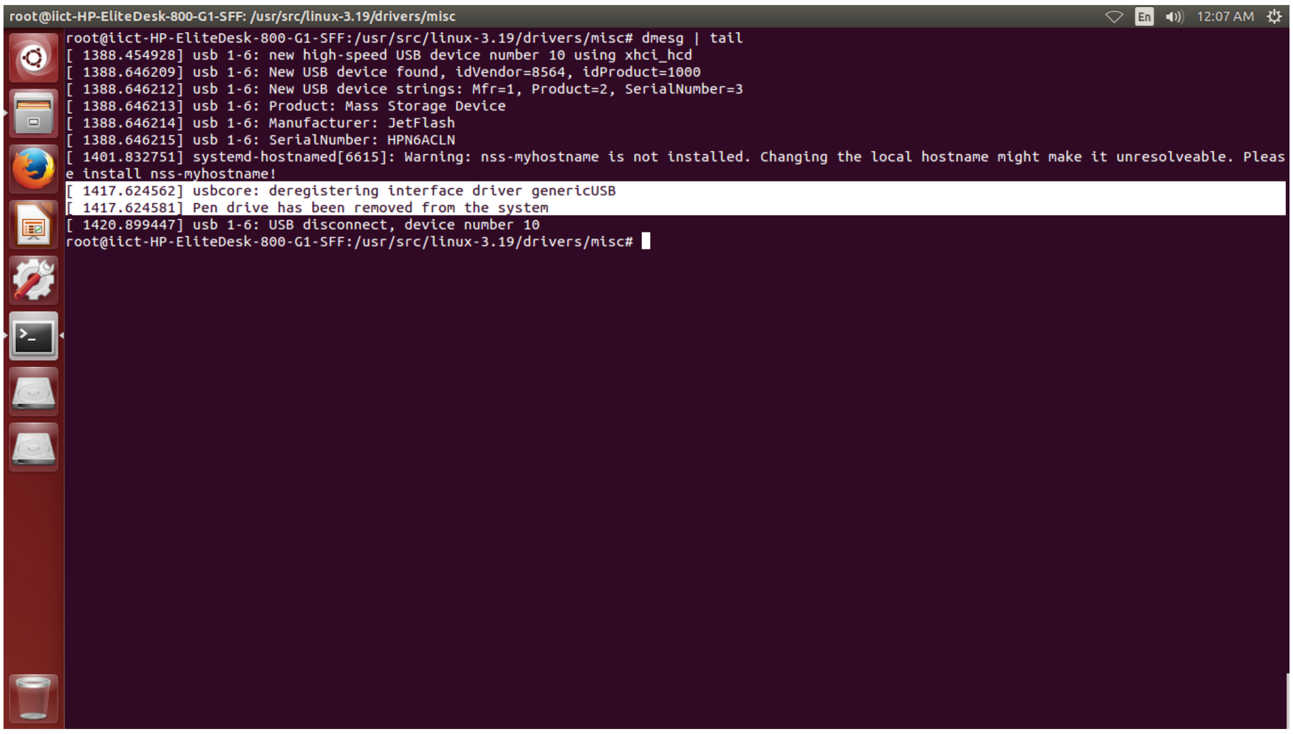
We implemented the USB module in kernel version 3.19.2



root@iict-HP-EliteDesk-800-G1-SFF: /usr/src/linux-3.19/drivers/misc
En
12:08 AM

```
root@iict-HP-EliteDesk-800-G1-SFF:/usr/src/linux-3.19/drivers/misc# insmod pendrive.ko
root@iict-HP-EliteDesk-800-G1-SFF:/usr/src/linux-3.19/drivers/misc# dmesg | tail
[ 1475.766980] usb 1-8: SerialNumber: HPN6ACLN
[ 1489.157883] usb 1-8: USB disconnect, device number 11
[ 1493.336445] Pen drive inserted into the system
[ 1493.336465] usbcore: registered new interface driver genericUSB
[ 1501.418581] usb 1-5: new high-speed USB device number 12 using xhci_hcd
[ 1501.609666] usb 1-5: New USB device found, idVendor=8564, idProduct=1000
[ 1501.609669] usb 1-5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1501.609670] usb 1-5: Product: Mass Storage Device
[ 1501.609671] usb 1-5: Manufacturer: JetFlash
[ 1501.609672] usb 1-5: SerialNumber: HPN6ACLN
root@iict-HP-EliteDesk-800-G1-SFF:/usr/src/linux-3.19/drivers/misc# 
```

Figure 1: Registering a new USB Mass Storage Device



root@iict-HP-EliteDesk-800-G1-SFF: /usr/src/linux-3.19/drivers/misc
En
12:07 AM

```
root@iict-HP-EliteDesk-800-G1-SFF:/usr/src/linux-3.19/drivers/misc# dmesg | tail
[ 1388.454928] usb 1-6: new high-speed USB device number 10 using xhci_hcd
[ 1388.646209] usb 1-6: New USB device found, idVendor=8564, idProduct=1000
[ 1388.646212] usb 1-6: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1388.646213] usb 1-6: Product: Mass Storage Device
[ 1388.646214] usb 1-6: Manufacturer: JetFlash
[ 1388.646215] usb 1-6: SerialNumber: HPN6ACLN
[ 1401.832751] systemd-hostnamed[6615]: Warning: nss-myhostname is not installed. Changing the local hostname might make it unresolveable. Pleas
e install nss-myhostname!
[ 1417.624562] usbcore: deregistering interface driver genericUSB
[ 1417.624581] Pen drive has been removed from the system
[ 1420.899447] usb 1-6: USB disconnect, device number 10
root@iict-HP-EliteDesk-800-G1-SFF:/usr/src/linux-3.19/drivers/misc# 
```

Figure 2: De-registering a USB Mass Storage Device

Conclusion

The kernel module we implemented required the existing USB module for mass storage devices connected to the system to be disabled to make the new custom module to work with the USB storage devices to interact with. The module works properly as seen in the implementation whenever a USB device register and deregisters with the Linux USB subsystem.

Further implementation part of the project included the data transfer between the USB storage devices and the Linux system. We did start to work on it but were not successful to complete it. We look forward to work upon this in future.

References

- http://matthias.vallentin.net/blog/2007/04/writing-a-linux-kernel-driver-for-an-unknown-usb-device/
- http://www.codeproject.com/Articles/112474/A-Simple-Driver-for-Linux-OS
- http://stackoverflow.com/questions/5911849/simple-kernel-module-for-usb