PRINTER DRIVER STUDY

Towson University

Document Revision 3

TEAM MEMBERS:

Table of Contents

Objective

To analyze, study, understand, and explain the functions of a USB 2.0 printer driver for the Linux (Ubuntu) operating system.

Linux supports a wide variety or peripheral devices, but characterizes them into three general categories char module, block module, or network module. A Universal Serial Bus or USB device is one way to connect devices and can span all three of those categories pending its purpose. Because of this USB devices have become the standard for connecting numerous devices to PC's. Ironically a USB is not a bus, but rather a tree built on point-to-point links. There is a host controller in charge of monitoring the device which is what allows for the plug and play simplicity of USB. In order to be so versatile USB had to create a standard for all devices to follow. Luckily, the Linux kernel can assist with handling the complexity of a USB driver because the kernel provides a system called the USB core. This research study will help identify and clarify the unseen interactions that occur within the operating system and the machines hardware for a printer device.

A printer as a printing device may be connected to a computer in the following ways;

a) USB port
b) Network connectivity
c) Bluetooth.

Introduction

One purpose of an operating system is to hide the eccentricity of the hardware from the user. In order to properly understand and develop device drivers for the Linux operating system you have to have a basic understanding of the following items:

- C Programming
- Microprocessor Programming

The Linux operating system (OS) has to work closely with the machines hardware and must acquire certain services that only the hardware can provide. The operating system handles processing these items through its Kernel.

The Kernel provides the basic services and device drivers used by all other programs running on a Linux OS system. Unlike the Windows operating systems where the kernel is proprietary the Linux kernel is free and is an open source software. This allows for a wider range of customization and support of external devices. Most users will never be aware of this kernel interaction because the operating system itself acts as an abstraction layer hiding this hardware

interaction. We plan to discuss this unseen interaction that occurs between the device, kernel, and operation system through this study. Before we can explain how the interactions occur and are processed we must understand what a kernel, a device driver, and USB connection are.

The Linux Kernel:

- Acts as intermediate layer to allow transition of information from the hardware layer to the operating system layer. Contains routines and functions that allow for interfacing between the operating system and the hardware.

Device Driver:

- Is software that handles or manages a hardware controller. Drivers are, essentially, a shared library of handling routines. There are many different types of drivers available, but they all consist of, kernel code, kernel interfaces, kernel mechanisms & services, loadable, configurable, and dynamic.

USB (Universal Serial Bus) 2.0:

- Allows high-speed, easy connection of peripherals to a PC. It has a maximum data transfer rate at 480 Mbps and found in over 6 billion electronic devices. Allows for 127 concurrent devices to be connected. Identifies its device data type to the host as either a interrupt, bulk, or isochronous.

For this project we will be analyzing the USB 2.0 connection with the Linux Ubuntu 10.4.x operating system. Our approach and implementation is documented below.

Implementation/Approach

1. Configuring a Linux operating system
    a. We will be using Ubuntu version 10.x
    b. Use image format (ISO) to install to a bootable device, hard drive, cd-rom, or flash drive .
    c. Run the Linux system

2. Download Source Code
    a. HP Printer Driver: http://www.hpdevelopersolutions.com
    b. Using the printer.c file located in the Hewlett-Packard Appliance Printing Development Kit (APDK)
    c. The printer.c code will be attached to then end of the research guide for review

3. Install USB Printer Driver
    a. Review installation interaction of printer device with the operating system
    b. Document USB interaction with OS
        i. Explain how the OS identifies the device and handles its association

        ii. Include screen shots from the OS about the device
    c. Printing process Overview
        i. Foomatic & PPD
        ii. CUPS

4. Print Test Page
    a. Analyze system calls and execution of printing
        i. Using Beagle USB Analyzer 480
        ii. Document screen shot interactions between printer and OS
    b. Analyze USB and OS data transfer
        i. Document interactions
        ii. Provide step by step interaction
5. Results
    a. Conclusions of USB printer drivers and Linux OS
    b. What we learned
    c. Future of USB device drivers

Install of USB Printer Device (OS Discovery)

Linux uses a module called Udev to monitor USB devices and assign them a node with a persistent device name. Prior to kernel 2.6 several different modules split the task, dev, devfs, sysfs and hotplug. Because these actions are run at each startup there is a significant benefit to reducing run time by initializing only those devices discovered on startup, and then monitoring as new devices are plugged in and removed.

Udev requires the use of rules to determine how, and who to assign names and drivers to devices. These rules are designed to be specific to each device; however there are broad rules to cover all devices for identification and provide some functionality to start.

As devices are discovered the kernel assigns them a device number. If there are two printers then depending on which printer is discovered first will determine the number that is assigned. To eliminate confusion, Udev rules identify the product information and associate it with the kernel number to assign it a persistent name.

The first time a device is attached Udev will create a rule for the particular device. The product description will be requested, and identified as a printer. This is where CUPS (Common Unix Printing System) comes into action. CUPS utilizes several database options such as Foomatic to determine the specifics and capabilities of the printer and then stores them within the Udev rule. The bulk of Udev and CUPS is maintained in user mode as opposed to Kernel mode, which allows for faster repair and addition of features, while separating what has to be handled by system calls to the Kernel avoiding kernel crashes with errors in the application. As devices become available or are removed Udev adds log entries to several log files:

- messages.log
- sys.log
- lpr.log

Below is an example of using the dmesg command in terminal and capturing the last 10 entries for and HP printer that was plugged into a USB port for a few seconds and then removed:

```
easy-2@easy-2-laptop:~$ dmesg | tail
[22990.018084] ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[23000.976517] wlan0: no IPv6 routers present
[32925.096143] usb 2-1: new high speed USB device using ehci_hcd and address 24
[32925.229317] usb 2-1: configuration #1 chosen from 1 choice
[32925.229564] hub 2-1:1.0: USB hub found
[32925.229735] hub 2-1:1.0: 2 ports detected
[32925.773726] usb 2-1.1: new high speed USB device using ehci_hcd and address 25
[32925.881827] usb 2-1.1: configuration #1 chosen from 1 choice
[33270.801785] usb 2-1: USB disconnect, address 24
[33270.801798] usb 2-1.1: USB disconnect, address 25
easy-2@easy-2-laptop:~$
```

Below is the rule that Udev uses to add printers:

```
# Low-level USB device add trigger
ACTION=="add", SUBSYSTEM=="usb", ATTR{bInterfaceClass}=="07",
ATTR{bInterfaceSubClass}=="01", RUN+="udev-configure-printer add %p"
# usblp device add trigger (needed when usblp is already loaded)
ACTION=="add", KERNEL=="lp*", RUN+="udev-configure-printer add %p"

# Low-level USB device remove trigger
ACTION=="remove", SUBSYSTEM=="usb",
ENV{ID_USB_INTERFACES}=="*:0701*:*", RUN+="udev-configure-printer remove
%p"
```

Below is a capture from the debug.log showing udev discovery and assignment. This action happens each time the printer is discovered:

```
4:37 easy-2-laptop udev-configure-printer: add /devices/pci0000:00/0000:00:04.1/usb2/2-1/2-1.1/2-1.1:1.0
4:37 easy-2-laptop udev-configure-printer: parent devpath is /devices/pci0000:00/0000:00:04.1/usb2/2-1/2-1.1
4:37 easy-2-laptop udev-configure-printer: Device vendor/product is 03F0:7704
4:37 easy-2-laptop udev-configure-printer: MFG:HP MDL:Deskjet D4100 series SERN:TH68J831K604D6
J831K604D6
```

```
4:40 easy-2-laptop udev-configure-printer: SERN fields match
4:40 easy-2-laptop udev-configure-printer: URI match: hp:/usb/Deskjet_D4100_series?serial=TH68J831K604D6
4:40 easy-2-laptop udev-configure-printer: SERN fields match
4:40 easy-2-laptop udev-configure-printer: URI match: usb://HP/Deskjet%20D4100%20series?serial=TH68J831K60
4:40 easy-2-laptop udev-configure-printer: URI of print queue: hp:/usb/Deskjet_D4100_series?serial=TH68J831K60
 deskjet d4100 series serial th68j831k604d6
4:40 easy-2-laptop udev-configure-printer: URI of detected printer: hp:/usb/Deskjet_D4100_series?serial=TH68J831
 deskjet d4100 series serial th68j831k604d6
4:40 easy-2-laptop udev-configure-printer: Queue ipp://localhost:631/printers/Deskjet-D4100-series has matching de
4:40 easy-2-laptop udev-configure-printer: URI of detected printer:
skjet%20D4100%20series?serial=TH68J831K604D6, normalized: deskjet d4100 series serial th68j831k604d6
4:40 easy-2-laptop udev-configure-printer: Queue ipp://localhost:631/printers/Deskjet-D4100-series has matching de
```

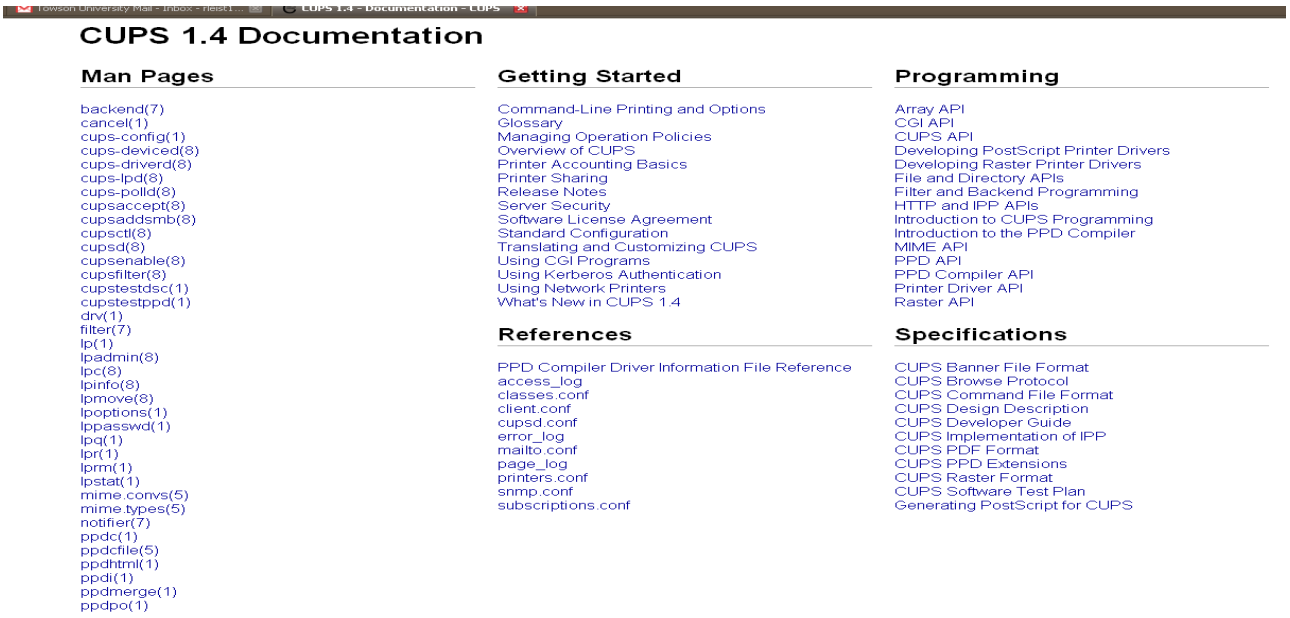
Foomatic and PPD OS libraries

Foomatic maintains what they claim to be, complete list of PPD (Postscript Printer Description) files for all printers. The database is referenced through CUPS using the URI of the printer to find the PPD. Each PPD file contains the specifics for each printer, including capabilities, dynamics, paper sizes, fonts, and styles.

CUPS

Common Unix Printing System was created by Apple to be able to successfully print using any printer with minimal setup, and with no need for internet connection to download additional drivers. Linux users quickly rallied behind the development, and as a result a local hosted printing system was created, and is now included with most distributions.

CUPS will handle print spooling, job monitoring, network, and local printers as well as the identification and further setup of the device. A large library of CUPS header files is available for system libraries, as well as API for individual characteristic of the printer. Postscript manipulation, image dithering, scaling, frame translations and rotations are just some of the CUPS filters, and API available.



CUPS 1.4 Documentation

Man Pages

backend(7)
cancel(1)
cups-config(1)
cups-deviced(8)
cups-driverd(8)
cups-lpd(8)
cups-polld(8)
cupsaccept(8)
cupsaddsmb(8)
cupsctl(8)
cupsd(8)
cupsenable(8)
cupsfilter(8)
cupstestdsc(1)
cupstestppd(1)
drv(1)
filter(7)
lp(1)
lpadmin(8)
lpc(8)
lpinfo(8)
lpmove(8)
lpoptions(1)
lppasswd(1)
lpq(1)
lpr(1)
lprm(1)
lpstat(1)
mime.convs(5)
mime.types(5)
notifier(7)
ppdc(1)
ppdcfile(5)
ppdhtml(1)
ppdi(1)
ppdmerge(1)
ppdpo(1)

Getting Started

Command-Line Printing and Options
Glossary
Managing Operation Policies
Overview of CUPS
Printer Accounting Basics
Printer Sharing
Release Notes
Server Security
Software License Agreement
Standard Configuration
Translating and Customizing CUPS
Using CGI Programs
Using Kerberos Authentication
Using Network Printers
What's New in CUPS 1.4

References

PPD Compiler Driver Information File Reference
access_log
classes.conf
client.conf
cupsd.conf
error_log
mailto.conf
page_log
printers.conf
snmp.conf
subscriptions.conf

Programming

Array API
CGI API
CUPS API
Developing PostScript Printer Drivers
Developing Raster Printer Drivers
File and Directory APIs
Filter and Backend Programming
HTTP and IPP APIs
Introduction to CUPS Programming
Introduction to the PPD Compiler
MIME API
PPD API
PPD Compiler API
Printer Driver API
Raster API

Specifications

CUPS Banner File Format
CUPS Browse Protocol
CUPS Command File Format
CUPS Design Description
CUPS Developer Guide
CUPS Implementation of IPP
CUPS PDF Format
CUPS PPD Extensions
CUPS Raster Format
CUPS Software Test Plan
Generating PostScript for CUPS

Printing Overview

Laser, Ink jet, and plotters have evolved and are not only able to print an image from a file, but can handle commands to dither, draw, and handle other image processing prior to printing, unlike their earlier predecessors. Postscript is a language that enables the use of vector graphics to print. The picture or object is described in terms of its characteristics such as circles, lines, points, and curves. The printer was then responsible to take the code and draw the image, and then print the image. Postscript is currently on its third version of the language where each level has added more functionality. This allows for scalability, and translations of objects in relation to the size of the paper.

However, PostScript requires that the printer have a processor to handle such a task. For those printers without a processor Ghostscript was created to compensate for printers that could not convert Postscript. Ghostscript uses the host computers processor to draw the image, and then allows the image to be passed to the printer as a bitmap. This is one of the filters available through many printing packages, such as CUPS for example.

The Beagle USB 480 Analyzer

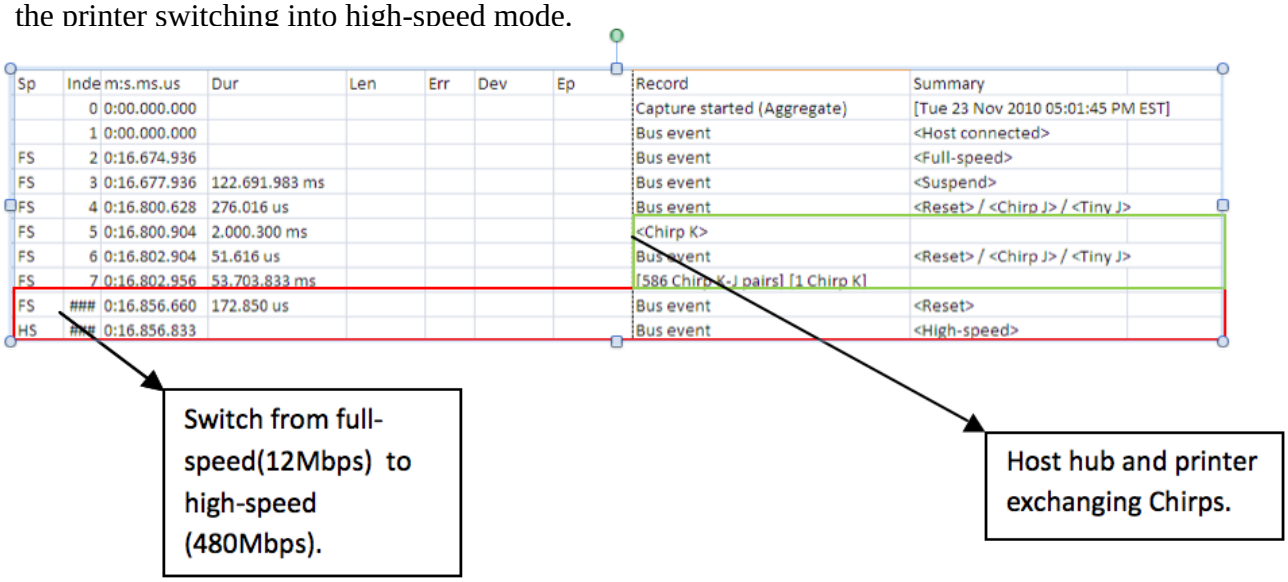
In order to interpret the code between the Linux operating system and the printer, a Beagle USB 480 Analyzer was used to view the messages sent back and forth between the printer and operating system. According to Total Phase (The manufacturer of the Beagle USB 480 Protocol Analyzer) the analyzer is used as a non-intrusive sniffer that gives engineers visibility into the bus traffic on their target device. These tools are used in a similar fashion to logic analyzers and oscilloscopes, but offer a higher-level view of the data. Using the USB

analyzer with the Beagle Protocol Analyzer Data Center software, the messages between the Linux operating system and the printer could be interpreted (Total Phase, 2007).

Operating System Interaction to Generate a Test Page

Initially, when a USB device is connected to a computer, the operating system determines if the device is a high-speed device. Until it can verify if the device is a high-speed (440 Mbps) device, the device runs in full speed mode (12 Mbps) (Compaq, 2000). In order to make the determination to switch the device into high-speed mode, the device and the operating system uses Chirps to determine the device‘s eligibility.

More specifically, the operating system and device use a differential voltage to detect if the hub and device are high-speed capable. “This procedure involves detecting the downstream chirp timed by a timer, which requires the device to perform the detection of the K-J-K-J-K-J chirps for at least 1.0 milliseconds, but at most 2.5 milliseconds. If the device is unable to detect a sufficient number of K-J transitions before the timer times out, the device enters the full-speed default state (Compaq, 2000)”. Reset ends when the bus state changes from SE0 to idle. In the illustration below we see 1) the operating system and the printer exchanging Chirps, 2) and then the printer switching into high-speed mode.



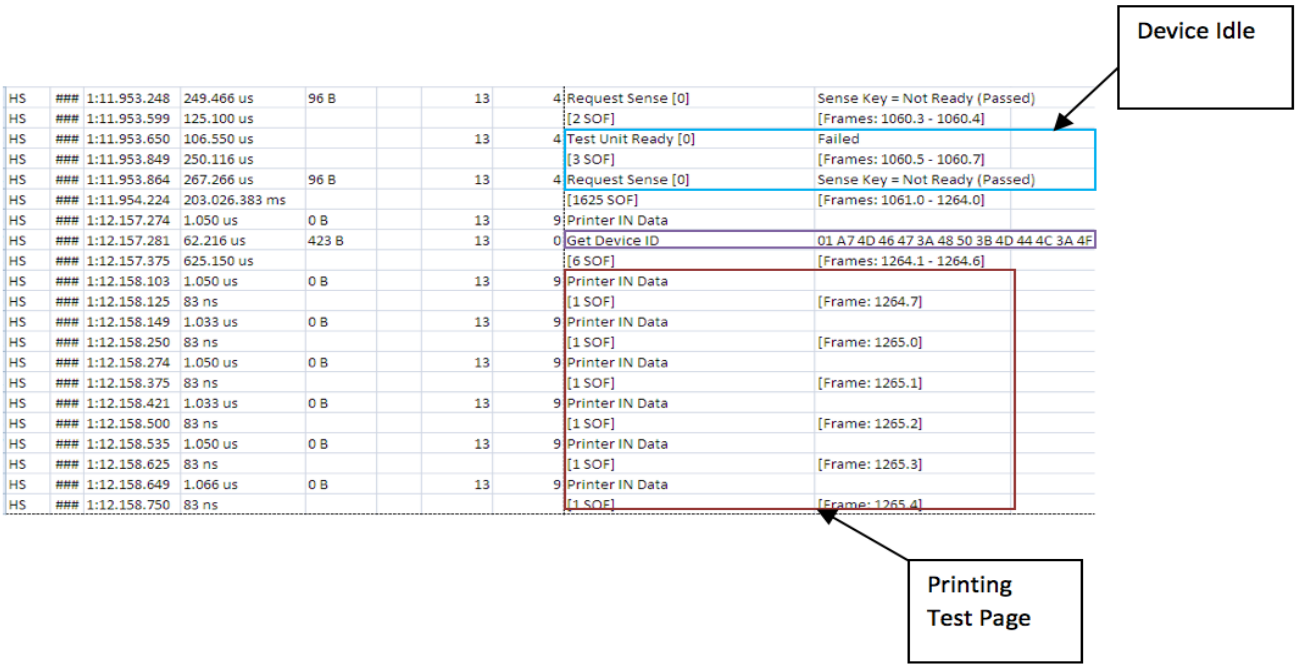
After the printer has been switched into high-speed mode, the operating system determines the printer’s capabilities. Moreover, requests are used to determine the capabilities. According to the “USB Device Class Definition for Printing Devices“ (Jan 2000) manual, a printer can respond to two different types of requests:

- Standard USB device requests, which perform general functions for supporting the bus and bus-related functions.
- Class-specific requests, which cause the device to transfer command data to or from the host.

The printer used in this demonstration used a combination of the two requests. The first request that was interpreted was the Get device Descriptor. "This descriptor contained the information that applies globally to the device such as the serial number, vendor ID, and the product ID. Using this information, the host Linux pc used this information to help determine what driver to load for the device (Total Phase, 2007)." The second, request that was analyzed was the Get Configuration Descriptor. According to the "Datasheet" by Total Phase, a device descriptor can have one or more configuration descriptors. Each of the descriptors can define how the device is powered (bus powered or self powered); the maximum power consumption, and what interfaces are available in a particular setup. The third request that was analyzed was the string descriptor. "This descriptor provides the host PC with human readable information about the device including strings for manufacturer name, product name, and serial number (Total Phase, 2007)". The fourth request, Set Interface, is a configuration descriptor that defines one or more interface descriptors. These interfaces are subdivided into multiple alternate interfaces that can finely modify the characteristics of the device (Total Phase, 2007. Finally, the last request Get Device ID, is a class-specific request that returns a device ID string (Phoenix Technologies, 2000)

HS	###	0:16.970.847					Bus event	<High-speed>
HS	###	0:16.970.847	53.757.033 ms				[431 SOF]	[Frames: 1380.x - 1434.4]
HS	###	0:17.024.668	3.966 us	0 B	0	0	Set Address	Address=13
HS	###	0:17.024.729	19.877.650 ms				[160 SOF]	[Frames: 1434.5 - 1454.4]
HS	###	0:17.044.668	34.333 us	18 B	13	0	Get Device Descriptor	ndex=0 Length=18
HS	###	0:17.044.732	83 ns				[1 SOF]	[Frame: 1454.5]
HS	###	0:17.044.764	14.316 us	9 B	13	0	Get Configuration Descriptor	ndex=0 Length=9
HS	###	0:17.044.857	83 ns				[1 SOF]	[Frame: 1454.6]
HS	###	0:17.044.880	19.033 us	124 B	13	0	Get Configuration Descriptor	ndex=0 Length=124
HS	###	0:17.044.982	83 ns				[1 SOF]	[Frame: 1454.7]
HS	###	0:17.045.014	14.400 us	4 B	13	0	Get String Descriptor	ndex=0 Length=255
HS	###	0:17.045.107	83 ns				[1 SOF]	[Frame: 1455.0]
HS	###	0:17.045.130	32.783 us	42 B	13	0	Get String Descriptor	ndex=2 Length=255
HS	###	0:17.045.232	83 ns				[1 SOF]	[Frame: 1455.1]
HS	###	0:17.045.258	18.433 us	6 B	13	0	Get String Descriptor	ndex=1 Length=255
HS	###	0:17.045.357	375.133 us				[4 SOF]	[Frames: 1455.2 - 1455.5]
HS	###	0:17.045.795	32.633 us	30 B	13	0	Get String Descriptor	ndex=3 Length=255
HS	###	0:17.045.857	250.116 us				[3 SOF]	[Frames: 1455.6 - 1456.0]
HS	###	0:17.046.095	21.550 us	0 B	13	0	Set Configuration	Configuration=1
HS	###	0:17.046.232	1.000.200 ms				[9 SOF]	[Frames: 1456.1 - 1457.1]
HS	###	0:17.047.330	15.566 us	0 B	13	0	Set Interface	Interface=1 Alt. Setting=0
HS	###	0:17.047.357	125.083 us				[2 SOF]	[Frames: 1457.2 - 1457.3]
HS	###	0:17.047.512	37.950 us	423 B	13	0	Get Device ID	01 A7 4D 46 47 3A 48 50 3B 4D 44 4C 3A 4F

After the printer was configured, it went into an idle mode until the user printed the test page. When the test page was initiated, the device interfaces were verified using the Get Device ID command. After, the interfaces were verified; the data was transferred from the host to the printer using the Printer IN Data command. After the test page was successfully printed, the printer switched back into an idle state until it was disconnected from the computer.



HS	###	1:11.953.248	249.466 us	96 B		13	4	Request Sense [0]	Sense Key = Not Ready (Passed)
HS	###	1:11.953.599	125.100 us					[2 SOF]	[Frames: 1060.3 - 1060.4]
HS	###	1:11.953.650	106.550 us			13	4	Test Unit Ready [0]	Failed
HS	###	1:11.953.849	250.116 us					[3 SOF]	[Frames: 1060.5 - 1060.7]
HS	###	1:11.953.864	267.266 us	96 B		13	4	Request Sense [0]	Sense Key = Not Ready (Passed)
HS	###	1:11.954.224	203.026.383 ms					[1625 SOF]	[Frames: 1061.0 - 1264.0]
HS	###	1:12.157.274	1.050 us	0 B		13	9	Printer IN Data	
HS	###	1:12.157.281	62.216 us	423 B		13	0	Get Device ID	01 A7 4D 46 47 3A 48 50 3B 4D 44 4C 3A 4F
HS	###	1:12.157.375	625.150 us					[6 SOF]	[Frames: 1264.1 - 1264.6]
HS	###	1:12.158.103	1.050 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.125	83 ns					[1 SOF]	[Frame: 1264.7]
HS	###	1:12.158.149	1.033 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.250	83 ns					[1 SOF]	[Frame: 1265.0]
HS	###	1:12.158.274	1.050 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.375	83 ns					[1 SOF]	[Frame: 1265.1]
HS	###	1:12.158.421	1.033 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.500	83 ns					[1 SOF]	[Frame: 1265.2]
HS	###	1:12.158.535	1.050 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.625	83 ns					[1 SOF]	[Frame: 1265.3]
HS	###	1:12.158.649	1.066 us	0 B		13	9	Printer IN Data	
HS	###	1:12.158.750	83 ns					[1 SOF]	[Frame: 1265.4]

Results

We discovered that HP printers from HPLIP require and have the following dependencies: Cups Library, Dbus, GhostScript, USB Library, Python Imaging, QT4, Python ReportLab, XSane, Network, Scan, Fax, and Cups drivers.

- **CUPS Library:** The CUPS library contains the entire core HTTP and IPP communications code as well as convenience functions for queuing print jobs, getting printer information, accessing resources via HTTP and IPP, and manipulating PPD files. The scheduler and all the commands, filters, and backends use this library.
- **Dbus:** D-Bus is a message bus system, a simple way for applications to talk to one another. In addition to interprocess communication, D-Bus helps coordinate process lifecycle; it makes it simple and reliable to code a "single instance" application or daemon, and to launch applications and daemons on demand when their services are needed.

  D-Bus supplies both a system daemon (for events such as "new hardware device added" or "printer queue changed") and a per-user-login-session daemon (for general IPC needs among user applications).

- **GhostScript:** Ghostscript is the name of a set of software that provides An interpreter for the PostScript language, with the ability to convert PostScript language files to many raster formats, view them on displays, and print them on printers that don't have PostScript language capability built in. It also provides A set of C procedures (the Ghostscript library) that implement the graphics and filtering capabilities that appear as primitive operations in the PostScript language and in PDF.

- **Python Imaging**: Is an external library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

Conclusion

In the old days a driver did everything including presenting the UI, communicating with the device and the translation task s. New modular print architectures break these tasks into specialized modules. In CUPS the “driver” only handles the translation task and a module called the “backend” handles the communication task.

Works Cited

1. Compaq, et al. 27 Apr. 2000. File last modified on 27 Apr. 2000. Universal Serial Bus Specification file.

2. "Connect Your Devices with udev." *Linux Format*. Wikipedia, n.d. Web. 8 Dec. 2010. <http://www.linuxformat.co.uk/wiki/index.php/Connect_your_devices_with_udev>.

3. Corbet, Johnathan, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers*. Ed. Andy Oram. Sebastopol: O'Reilly Media, 2005. *LWN.net*. Web. 8 Dec. 2010. <http://lwn.net/Kernel/LDD3/>.

4."CUPS 1.4 Documentation." *Cups*. N.p., 2010. Web. 8 Dec. 2010. <http://www.cups.org/documentation.php>.

5. "Driver Core: devtmpfs - kernel-maintained tmpfs-based /dev." *Linux Info from the Source*. N.p., n.d. Web. 8 Dec. 2010. <http://lwn.net/Articles/345480/>.

6. "Linux and USB 2.0." *Linux-USB*. SourceForge.net, n.d. Web. 8 Dec. 2010. <http://www.linux-usb.org/>.

7. Phoenix Technologies. "Universal Serial Bus Device Class Definition for Printing Devices." *USB*. USB Implementers Forum, 2000. Web. 6 Dec. 2010. <http://www.usb.org/developers/devclass_docs/usbprint11.pdf>.

8. Total Phase Inc. "Beagle Protocol Analyzer Data Sheet v4.00." *Total Phase*. N.p., 2010. Web. 6 Dec. 2010. <http://www.totalphase.com/docs/beagle_datasheet/contents/>.

9. Ubuntu. "Printing." *Ubuntu*. N.p., n.d. Web. 8 Dec. 2010. <https://help.ubuntu.com/10.04/printing/C/printing.html#local>.

10. Software dBus. "What is D-Bus?" 12 Dec. 2010<http://www.freedesktop.org/wiki/Software/dbus>.