OS PROJECT

Thermal Printer Driver

Submitted by,

Emmanuel Scaria - B140241CS

Navaneeth Kishore - B140143CS

Nithin Jose Tom - B140090CS

The project we decided to work on is a driver for a small thermal printer that is print receipts and bills. The reason we decided to make such a driver is because a thermal printer is being used in the institute’s digital library under windows and since a linux driver isn’t available, one system is still running windows even though the plan was to make the entire digital library run on linux. We decided to make this driver so we could run the printer on linux and make the digital library independent of proprietary software. Since Navaneeth is the student library admin, we were able to get access to the system connected to the printer and work on it remotely.

The first thing we worked on was finding out what kind of device we were dealing with and how printing was done in linux. On googling the model, we found the instruction manual and the command manual but couldn’t make sense of them since they were made for professional developers. We found that applications send the print job as a PDF file to the printing system and the printing system (CUPS) converts the job to the printer's language. The resulting data stream it sends through a printer port to the printer. It was only through a lot of trial and error after going through the CUPS (Common Unix Printing System) source code, that we figured out that communication to the printer (which is a USB device) could be done using the libusb library in the user space.

We learned about USB interfaces and endpoints. A USB device defines a group of end-points, where are grouped together into an interface. An end-point can be either "IN" or "OUT", and sends data in one direction only. End-points can have a number of different types: Control, interrupt, bulk and isochronous. There can be many interfaces (made of multiple end-points) and interfaces are grouped into "configurations". Most devices only have a single

configuration. We also figured out that out of the four modes of communication with usb devices, our device uses bulk communication.

Next we did research on the libusb library to get started with communicating with the printer. On figuring out how basic libusb commands and functions work, we tried to communicate with the device with a small sample file. Through the commmand manual, after a lot of reasearch, we were able to understand how we could use the printer commands in our user space code. There are a variety of commands for each operation and these commands are all basically a string of numbers (hexadecimal). There are commands for printing, checking the status, changing printing position etc.

Upon close inspection of the device (VendorID: 3540 ProductID: 428), we found that it has just one interface, and that interface has two enpoints 129 (OUT) and 2 (IN) (through “sudo cat /sys/kernel/debug/usb/devices”). At this point we were able to control the printer from user mode. On figuring out how all this worked, we were able to make a user space program to successfully print characters using the libusb library.

The libusb library allows us to manipulate usb devices. We used the libusb_bulk_transfer function to send the required characters into the printer buffer and the commands for printing. The other functions we used were to initialise the device, claim the interface while working on it (like a mutex) and other trivial functions.

Next, we moved on to working with the kernel. We learned about making a kernel module while looking up how to get device drivers to work. Since kernel modules could be loaded and unloaded dynamically, we didin’t have to recompile our entire kernel each time and could work on our own systems without much risk.

The constructor and destructor for our kernel module had the commands for registering and deregistering the printer. We assigned functions for probing and disconnecting the device. When the printer is connected, the USB core subsystem searches for a driver that matches with the printer's vendor id and product id. When it finds our driver, it invokes the probe function associated with our driver. The probe and disconnect functions registered the deregistered the interface we work with.

For actually printing data, we make use of the read and write properties of device files. We make a device file for the printer and send data to it for it to be printed. This is handled by the read and write functions we have created and associated with the device file. If we try to read data from the printer, we try getting data from its buffer which may not even be meaningful. The data we send is transferred in bulk format to the printer along with the instructions to initialise the printer and cut the paper after printing. When the printer is disconnected, the disconnect function is called and the printer is deregistered.

The main sources we referred for information were:

1.) https://github.com/apple/cups
2.) http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html
3.) http://opensourceforu.com/2011/12/data-transfers-to-from-usb-devices/
4.) www.libusb.org
5.) http://www.usb.org/developers/docs/
6.) http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/