

Printing from Embedded Systems

Last Updated Friday, 12 December 2008

How does one implement support for printing in embedded systems? I recently had the opportunity to add printing support to an embedded Linux system. The device is an industrial touch screen powered by a Compulab cm-x270 module (PXA270 CPU), and runs a GTK+ application. The customer is implementing a device calibration system where customers bring their equipment in to get calibrated, and the system prints out a report on a local printer. This article describes how components of Hewlett Packard's HPLIP solution along with Cairo can be used to implement printer support in a non-desktop Linux system.

Requirements

The requirements for this project were fairly basic -- we needed to print a single page report that contained text and some elementary graphics. We initially wanted to support several low-end Inkjet and Laser USB printers. As the system was powered by a fairly slow ARM processor (slow compared to modern desktop systems), the solution needed to be efficient, and not require excessive amounts of memory. Desktop Linux Print Flow

With desktop Linux systems, the standard printing flow looks something like this:

Application -> PS output -> Ghostscript -> Rasterized output -> Printer Driver (filter) -> I/O Backend -> Printer

CUPS is typically used to manage this flow, provide queuing, etc. Print Architecture

As the application is written in GTK+, we decided to generate the report using Cairo, which is a 2D graphics library. Cairo is easy to use, and is well suited for this application. With a Cairo generated report, we already had a raster image of the report, so it seemed that the PS output/Ghostscript steps were not really needed and only added more processing to the data flow. Also the PXA270 CPU does not have a FPU, and PostScript processing can be floating point intensive. So the need was now to figure out how to get a Cairo generated raster image to a USB printer.

Hewlett Packard offers lots of interesting software for their printers. Their APDK is a OS independent library in source code form. However, implementing this would have required quite a bit of integration, and writing the I/O layer. It seemed like there should be something available that would work with a little less effort. The HPLIP project is a comprehensive set of software for Linux printing, and is used by desktop Linux systems. It is not obvious at a glance how all the components of HPLIP fit together, but after spending some time digging through source code, and asking questions on several maillists/forums, we were able to figure out that the basic flow is:

Ghostscript -> HPIJS (driver) -> HP backend -> Printer

In this application, we simply launched the HPIJS driver directly from our application instead of Ghostscript. HPLIP

There were several challenges using HPLIP in our ARM system. The HPLIP is not cross-compilation friendly out of the box, so we had to fix a few issues, and ended up disabling all the Python pieces as we only needed the HPIJS and backend components. We also had to figure out the data flow from the application to HPIJS, and then to HP backend. In a nutshell, the process is:

- start the hp backend process by forking, and obtain a file handle to STDIN for the hp backend process
- start the HPIJS process, send various parameters to it including the file handle for the backend STDIN
- send HPIJS the print raster and tell it to print

There are some additional details such as handling margins, error conditions, cancel support, etc. Some of the details can be gleaned from the Ghostscript source code, and the IJS reference implementation provides some very useful library code for implementing the IJS client functionality in the application that is doing the printing. When finished, our printing module was 403 source lines of code (SLOC) -- not bad considering the functionality. How well does it work?

Overall, we are very pleased with the result. With a simple industrial terminal, we can now support just about every USB printer made by HP with the exception of some of their very low end LaserJet printers that require a binary plugin. The HP backend can be used to detect what printer is attached at run time, so everything is plug-n-play with no user configuration. Let me repeat as this is significant -- on a simple Industrial terminal, we can support about every HP USB

printer available with no user configuration. This is easier for users than their desktop as they simply need to buy a printer and plug it in! Kudos to HP for their excellent open source software. Because the source code is available, we were able to customize it for our application with very little support from HP. Performance wise, the printing process is quite fast; the printer starts almost immediately after the user initiates the print. Now if I could just get HP to build a ARM version of their binary plugin for the few low end lasers we can't support ... but this is not a critical issue for this vertical application. Future Direction

Hopefully, this type of solution can evolve into a standard printing solution for Embedded Linux systems. Epson also supports the IJS driver model, so adding support for their printers should be possible. It may eventually make sense to integrate portions of CUPS for queing, and other management tasks. Some of the tasks I hope to implement in the future:

- Clean up the HPLIP build for inclusion in OpenEmbedded.
- Clean up the IJS reference code library build and packaging in OpenEmbedded.
- Figure out portions of CUPS that may make sense.
- Keep conversations going with the Linux printing group so we can someday have a "standard" printing solution for embedded Linux systems.

Thanks to Matt Gessner for helping implement this solution, and for providing feedback on this article.