Subject: [Printing-architecture] OP Thin Thread Developement for LinuxEmbedded Solutions

Attachments: ATT816421.txt

All,

As discussed at the last SC and Arch meeting one of the possibilities for continued activities within the Open Printing in the US/Europe is to develop a thin-thread implementation of the Open Printing Architecture for a Linux Embedded Solutions. I have been asked to pull together my thoughts and ideas for such a development to get things started.

For embedded solutions, the implementation limitations are typically,

1. Limited memory
2. Slower processor speeds
3. May or may not have a file system
4. May or may not have persistent memory (recoverable after power off/on)
5. The solution my use a reduced operating system configuration
    a. May imply no system level memory management
    b. Again, no file system
    c. Unidirectional communication (I/O) configuration

The following are my suggestions the thin thread implementation of the current open-printing architecture.

Printer I/O will only be USB - Bidirectional capable.

"op" meaning Open-Printing will be used as the prefix in my comments below

1. There will need to be some common components; namely,
    a. Pointer-to-functions for
        i. Memory Functions
            1. opMalloc, opReAlloc, opFree
        ii. File Functions
            1. opOpenFile, opReadFile, opWriteFile, opCloseFile
        iii. I/O Function - Low level USB I/O for use within the Print Channel Manager
            1. opOpenUSB, opReadUSB, opWriteUSB, opCloseUSB
    b. Generic Error Messages
        i. "Out of Memory", "Invalid Value", "Internal Error", ...etc
    c. Generic Type Definitions
        i. Basic types: opBIT8, opINT8, opBIT16,.....etc
    d. Generic structures
        i. Image-struct, job-struct, ... etc.

2. Application:
    a. Since the application is not important and, actually, the content is not important. We need an application that prints something; there the application suggested is a simple image application that will print 4x6 photos. Thus, the application needs to,

        i. Open an image file
        ii. Read image data into memory.
        iii. Display the image on the screen.
        iv. Obtain the current printers capabilities
        v. Display printer capabilities and settable options
        vi. Input / Change settable options
        vii. Scale/Clip/Rotate image as necessary to fit on a 4x6 page.

viii. Initiate a Print Job
    1. By direct interface with the Print Manager
    2. By interface with Job Ticketing
ix. Print the image in bands
x. Close the Print Job

3. Print Manager
    a. The current plan is to utilize the existing PAPI interface. Due to the implementation constraints of embedded solution, it may be necessary to reduce the set of or functionality of API within PAPI. If this is necessary, then caution must be taken to ensure that scalability of PAPI is not lost.

        i. Implementation details will have to be discussed.

4. Job Ticketing
    a. The job ticketing will not be able to use a known job ticket like JDF; I believe it will be too complex. Therefore, I suggest the job ticket directly support the OP/JTAPI defined elements. Further, I suggest that the internal structures be defined that directly support the OP/JTAPI defined elements and that these structures are used through out the entire architecture.

5. Spooler
    a. For an embedded solution, the spooler is basically a very limited data spooler or temporary memory buffer for printing data.

        i. It is important, even at this limited capability, to instantiate a scalable set of API's

6. Transform
    a. Transform can not include GhostScript or the equivalent post-script interpreter; it is simply too big for any real embedded solution. Therefore, I suggest that the thin thread implementation have a few simple transform functions

        i. Convert Color to Grey
        ii. Rotate Image 180 degrees
        iii. Scale (up and crop) image

7. Device Discovery
    a. A simple polling of the USB interface could be implemented; where the printer USB identifier string is retrieve. [The identifier is then used to retrieve a PPD file or equivalent printer description file by the Capability Module.]

        i. [optional] Poll the USB interface to retrieve the Printer's USB Identifier
        ii. [optional] Store the Printer's Identifier
        iii. Notify the Print Manager of the printer
        iv. Notify the Capability Module of the new printer

8. Capability Module
    a. The PCAPI will retrieve the printer's specific information and put the information in term the current solution can utilize.

        i. Retrieve the Printer's description file based on the Printer's Identifier.
            1. [Retrieve info directly from the printer will be considered out of scope for the thin thread activity.]
        ii. [??? Interpret data for the current solution.
            1. What capabilities are supported by the application
            2. How are the capabilities supported by the application
        iii. Notify the Printer Manager of the Printer's Capabilities]

9. Status Monitoring
    a. Although the API for this module is open source, the actual implementation, in general, will be printer vendor specific and, as such, may or may not be public. Therefore, the recommendation here is to request one of the printer vendors to provide a compiled implementation. The OP architecture team should investigate at what level it is necessary to support proprietary software and how. Example, can an IJS-like status monitoring interface be defined.

        i. Details to be worked out.

10. Driver
    a. This is same situation as Status Monitoring I would make the same recommendation. In this case the Vector-Driver will be used, in part or whole, as the basis for a solution.

        i. Details to be worked out.

11. Print Channel Manager
    a. The PCM must be implemented as originally conceived; that is, a vendor independent I/O manager. The recommendation of the SC and Arch teams is that the initial version only supports USB. To support status monitoring and the capabilities function the USB I/O must be bidirectional.

Next Steps

1. We must determine "How this work will be done/" before proceeding.

Assuming we can defined a realistic "how to do this work", then I believe the next couple of steps are

2. Refine the above outline to get consensus from the contributors
3. Assigns each modules discussed above to a specific individual
4. Agree on the development environment
    a. As silly as it might seem, I actually do this in Window (using Visual C++ 6.0). The final code is moved to Linux and confirmed to compile in run. Using this approach general makes the code platform/solution independent.
5. Begin development of individual modules.

Rgds,
Glen W. Petrie
Epson Imaging Technology Center
2580 Orchard Parkway, Suite 200
San Jose, CA, 95131
Voice: 408.576.4131 Fax: 408.474.0511