A Technique for Automatically Reprogramming an Embedded Linux System

Gianluca Valentino
Department of Communications
and Computer Engineering
University of Malta
Msida 2080, Malta
Email: gval0001@um.edu.mt

Saviour Zammit
Department of Communications
and Computer Engineering
University of Malta
Msida 2080, Malta
Email: saviour.zammit@um.edu.mt

***Abstract*—This paper presents a method used to automatically reprogram an embedded Linux system using a USB (Universal Serial Bus) pen drive. New software stored on the pen drive can be automatically transferred and installed into the embedded Linux System on insertion of the pen drive. The technique was developed with the aim of creating a multi-purpose device that could be easily reprogrammed at will. This device would then allow a user to bypass a computer in order to transfer data between USB peripherals.**

***Index Terms*—USB pen drive, Automatic reprogrammability, Embedded Linux.**

I. Introduction

The method of automatic reprogramming was developed as part of the implementation of an Intelligent Peripheral Controller (IPC) [1]. This reprogrammable device is capable of automatically detecting USB peripherals on insertion and performing various tasks accordingly. Examples include the automatic transfer of data between pen drives or the automatic printing of a file located on a pen drive. The IPC would be automatically reprogrammed by inserting a USB pen drive containing the new software to be installed.

Existing methods for automatically reprogramming embedded systems tend to consider the updating of low-level code (firmware), rather than high-level programs. Yao et al. [2] maintain that as embedded devices are achieving network connectivity, online-update of their software is becoming more of an issue. They present an extended form of the normal boot loader. The new software is obtained from a TFTP server, and is installed by writing it to flash memory.

The risks involved in updating embedded system software are discussed in [3]. In critical systems that have to run continuously, stopping the system for an update is not the ideal way to do things. The contribution made in [3] is the design of a software framework in which an application program can be updated 'dynamically' while running.

Li et al. have applied the automounting technique to develop an automatic software install/update system for Linux [4]. They placed an executable shell script file into a RPM package, which in turn was transferred to a USB Mass Storage device. When the device was inserted into the embedded system, it was mounted automatically, and the shell script file was executed. The disadvantage of their approach is that their implementation can only detect and automatically install software from a pen drive if only one such device is connected. The automatic software installation procedure that they claim is quasi-automatic, since the name of the new software package is actually hard-coded into the software and thus the procedure cannot be used to install software packages with different names without modifiying the software each time.

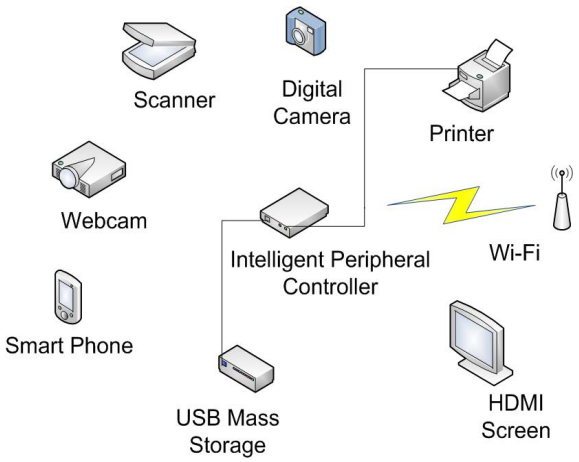

Fig. 1. The various peripherals that can be interconnected via the IPC

The concept of the Intelligent Peripheral Controller (IPC) shown in Figure 1 and developed in [1] is that of a reprogrammable device that can detect connected USB peripherals and automatically perform an action accordingly, without requiring the intervention of a Personal Computer. The example illustrated in Figure 1 shows the connection of a USB pen drive to a USB printer, thus allowing a user to print a file located on the pen drive on connection of both devices to the IPC. Other examples include the transferring of data between two smart phones, or the transferring of data located on a digital camera to a USB pen drive. The IPC can be automatically reprogrammed or the amount of tasks it can perform can be increased by inserting a USB pen drive containing the new software to be installed.

The automatic techniques reported in the literature and

discussed above do not fulfil the criteria of a user-friendly plug-and-play system, thereby motivating this work. A detailed description of features involving the data transfers between USB peripherals mediated by the IPC is beyond the scope of this paper, and can be found in [1].

This paper is organized into three sections. Section II discusses the design parameters and hardware/software issues involved. An overview of the technique implementation is given in Section III, and evaluation and testing results are provided in Section IV.

II. Design Considerations

A. Hardware

The automatic reprogrammable mechanism was implemented and tested on the BeagleBoard [5]. The selection of this development board was made because of its popular usage in prototyping embedded system applications. Its features include an OMAP 3530 ARM Cortex-A8 based microprocessor (720 MHz), 256 MB of RAM as well as support for a substantial number of peripherals, such as USB, HDMI, SD Card and audio devices.

B. Software

Ubuntu was chosen as an embedded Linux OS because of its wide support for the BeagleBoard. A kernel and a root file system were built for the Ubuntu 9.10 distribution using rootstock [6]. The C language is ideal for implementation of the technique since it results in efficient, portable code, and can easily provide access to the underlying hardware [7].

III. Implementation of the Technique

A. USB Pen Drive Detection Mechanism

A peripheral insertion detection mechanism is required in order to determine whether a pen drive has been inserted into the system. One such mechanism is udev [8]. Its features include the ability to execute programs when certain device events occur (such as insertion or removal), as well as allowing access to information about currently attached devices. Thus, the automatic reprogramming software resident in the permanent memory on the board would not need to continuously poll the device for any activity.

B. Program Implementation

In order to keep production costs low, the current implementation assumes a device (such as the IPC) that is devoid of any user interface, such as a screen. Since this denies the user the option of manually selecting the new software to be installed, the user is therefore required to place the new software in a certain pre-established folder on the pen drive, for example 'New Software'. The insertion of the pen drive triggers udev, which in turn runs the program (written in C).

This program, termed *auto-install*, searches for and copies any new software packages on the pen drive to the /usr/local/packages directory on the board. The software is determined to be 'new' by checking whether a software package of the same name already exists in the directory. The software package consists of two files, namely the executable binary and an installer script.

The executable binary is extracted to /usr/local/bin, and is run whenever the new feature is required, for example the feature of automatically transferring files between two pen drives. The installer script is extracted to /usr/local/install, and is executed to update the udev rules file with new rules. This ensures that the new devices to be supported by this feature will be detected on insertion, and the corresponding executable will be run by the udev rule. If one considers the new feature to be that of being able to print directly from a pen drive, then the new devices are the pen drive and the printer. Extraction of both the executable binary and the installer script is performed by copying the file with the standard C *fread* function.

The inner loop that checks whether the file name length is greater than 7 ensures that the current directory and parent directory files are not processed. The automatic reprogramming technique loops for as long as there are new software packages present on the pen drive. A graphical depiction of the process is shown in the flowchart in Figure 2.

IV. Evaluation and Testing Results

The evaluation and testing of the technique consisted of observing whether new software updates stored on the pen drive are correctly installed. In addition, a set of benchmarks were developed to compare the speed of execution of the automatic reprogramming technique when running on the BeagleBoard with the same program re-compiled and running on a MSI EX600 laptop [9], also running Ubuntu 9.10. The purpose of the latter test was to evaluate the reduction in performance, if any, of the embedded system against a fully functional PC-based Linux system.

A. Verification of the Technique

A typical scenario consisting of a new feature supporting the automatic transfer of data between two pen drives was considered. On insertion of the pen drive with the 'usb-to-usb' zip file located in the 'New Software directory on the pen drive, the executable binary *usb-to-usb* and the installer script *usb-to-usb_install* were verified to have been copied to the appropriate locations by looking at the directory structure on the board via a minicom terminal from the host computer.

The udev rules file was also verified to have been updated with the following rules:

```
ACTION==“add”, KERNEL==“sd[a-z]*”,
RUN+=“/usr/local/bin/usb-to-usb %k 1”
ACTION==“remove”, KERNEL==“sd[a-z]*”,
RUN+=“/usr/local/bin/usb-to-usb %k 2”
```

The rule is therefore sensitive to changes in '/dev/sd[a-z]*', which is a regular expression indicating files in /dev associated with USB pen drives. If a pen drive is added to the system, the *usb-to-usb* program is executed. Parameters may also be passed to the program. In the above example, '%k' refers to the kernel name (e.g. sda, sda1 etc.) while '1' is a flag specific to this example, indicating that the pen drive has been inserted.

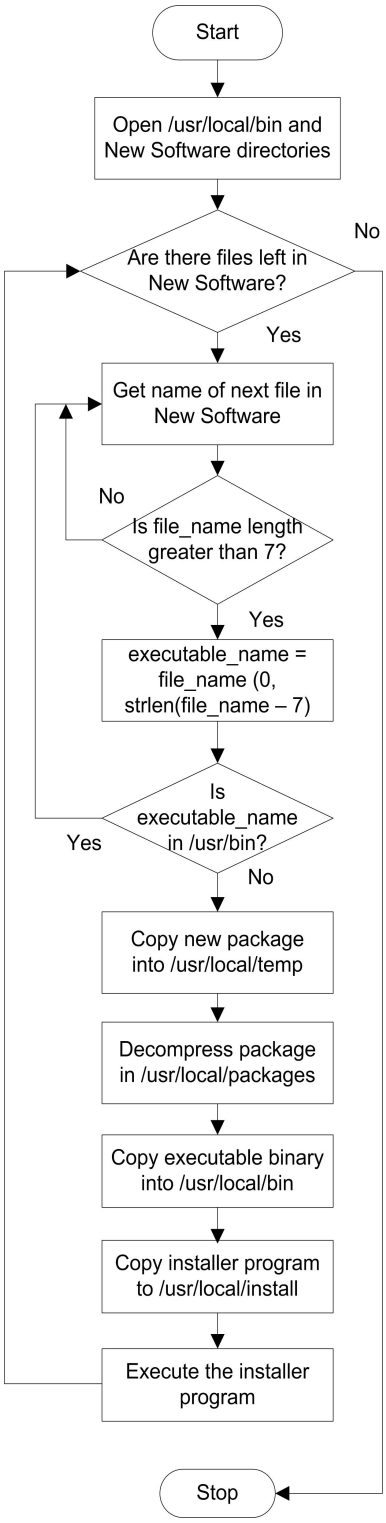

Fig. 2. Flowchart of Program Execution

TABLE I
SOFTWARE PACKAGES USED IN TESTING

Software Package	File Size (KB)
usb-backup	15.8
camera-backup	13.2
usb-to-usb	43.4
webcam-display	12.1
usb-to-printer	39.6
printer-config	45.8
wifi	14.1
wifi-config	44.5

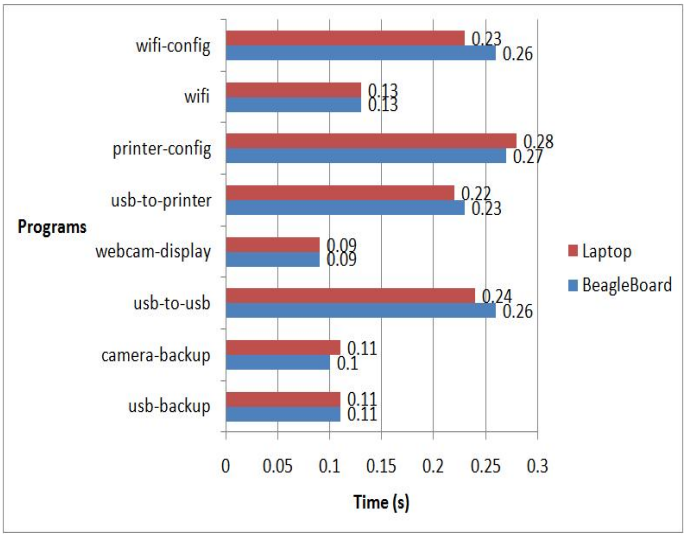

Fig. 3. Comparison of auto-install Execution Duration

In the case of the second rule, '2' is a flag that indicates that the pen drive has been removed.

B. Performance Analysis of the Technique

The time taken for the automatic reprogramming of the IPC to occur for a given feature was measured using *time*. This is a standard Linux utility used to determine the time taken for a program to execute. The MSI EX600 laptop supports a 2 GHz Intel ® Core ™ 2 Duo processor and 2 GB of DDR2 RAM. A comparison of the times taken to install various programs whose implementation is explained in [1] is shown in Table 1.

The comparison in execution duration illustrated in Figure 3 was made to establish whether it would be feasible from a view-point of speed to implement the software technique on a low-cost device which could support multiple features. The results obtained show that the execution duration for the automatic reprogramming technique when running on both types of hardware is comparable. In addition, the time taken to install the a new program is proportional to its size, which is as expected. One must consider that these results were obtained notwithstanding differences of approximately a factor of 8 in the amount of RAM and a factor of 2.8 in the CPU speed of the two types of hardware.

V. Conclusion

The contribution of this work was the design and implementation of a method to automatically reprogram an embedded Linux system with the use of a USB pen drive. The current implementation runs a binary executable on top of a Ubuntu Linux operating system, together with udev as a hardware detection mechanism. Although Ubuntu was used as an embedded Linux System, the technique can be extended to any other distribution provided that udev is installed. The software technique was verified to function correctly, and its performance in terms of execution speed was found to be comparable to the same implementation running on a MSI EX600 laptop. The results of this research show that it is feasible to implement such a technique on a small device, which could then be used to bypass a desktop computer in performing daily tasks involving the transferring of data between USB peripherals.

References

[1] G. Valentino and S. Zammit, *Design and Implementation of an Intelligent USB Peripheral Controller*, presented at the 3rd National ICT conference, *WICT 2010, San Gwann Malta*.

[2] G. Yao, W. Zhang and J. Wang, *The design and implementation of online-update on embedded devices*, in Proc. of IEEE International Conference on Computer Science and Software Engineeering, 2008, pp. 24-27.

[3] M. Hicks and S. Nettles, *Dynamic software updating*, in ACM Transactions on Programming Languages and Systems, vol. 27, no. 6, November 2005, pp. 1049-1096.

[4] T. Li and H. Pei-wei, *Automatic software install/update for embedded linux*, in Journal of Shanghai Jiaotong University (Science), vol. 13, no. 1, February 2008, pp. 107-109.

[5] BeagleBoard System Reference Manual Rev C4, December 2009.

[6] Rootstock project web page. [Online]. Available: https://launchpad.net/projectrootstock

[7] M. Barr and A. Massa, *Programming Embedded Systems with C and GNU Development Tools*, 2nd edition, O'Reilly, 2006.

[8] G. Kroah-Hartman, *udev - a userspace implementation of devfs*, in Proceedings of the Linux Symposium, July 2003, pp. 263-271.

[9] MSI EX600 Manual, June 2007.