

Fast learning methodology for GNU/Linux driver development

B. Caballero¹, J. Rodriguez¹, C. B. Navarrete^{1,2} and E. Anguiano^{1,2}

¹Centro de Referencia Linux UAM-IBM, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain.

²Dpto. Ingeniería Informática. Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain.

It is pretended to present an effective teaching system for programming of Linux device drivers is presented. With this system the basic skills to create Linux device drivers can be obtained in efficient manner during one hour of theoretical lesson and two hours of workshop with and hands on. In order to get the best learning results it's strongly recommended to use real hardware devices. The most of the current devices are too complex to learn in an easy way. For this reason it has been decided to create a low complexity and low cost hardware device, in order the students can practice with it, without affecting the rest of the system. The student will have a suitable peripheral for learning, that will allow they to have an absolute control of all the communication with the computer.

Keywords device drivers; GNU; Linux; fast learning

1. Introduction

During the Summer courses of the Universidad Autónoma de Madrid (Spain) in July 2006, the Linux Center Research (CRL, UAM-IBM) organized a practical and theoretical session of 3 hours, oriented to understand the procedure and implementation of a GNU/Linux [1] driver. In order to simplify the process, the device connected to the parallel port [2] was used, that was adapted for the educational technics to be teach to the students.

This paper is divided in three sections: the first section presents an introduction of the model and the methodology to implement the Linux driver; in the second section we introduce the theoretical aspects with a short description to the GNU/Linux system and the way that this operating system manage the devices connected to the parallel port. In the last section we expose the methodology and the steps needed to implement the driver. At the end of this workshop, we present the conclusions based on the experiences and results of the programming exercises performed by the students.

2. Designing the hardware device

In order to explain the concept of driver, a perspective of a physical versus logical device is approached. The physical device is the hardware that is connected to the computer. Instead of this, the logical device is the space defined by the operating system used to carry out the special commands to the physical device. The driver is the communication element between logical and physical layer.

Approximately all of the actual devices are too complex structure for learning driver programming; that is the reason why we have decided to create an adapted device to our needs. This device fulfils a series of requirements adapted to our purpose: simplicity, easy using and also low cost. We have take the following design decisions:

1. Use of a parallel port due to its simplicity and its big knowledge. Simultaneously this port offers rapidity and a good well known properties.

- Use of a set of elements of low cost for the communication with the computer: leds and 8-display segments to show the required information.

Before the summer course, some schematics to specify the connections needed to the correct operation of the device where designed (fig 1.) by the staff of the CRL group. This schematic was physically implemented and given it to each student registered in the course.

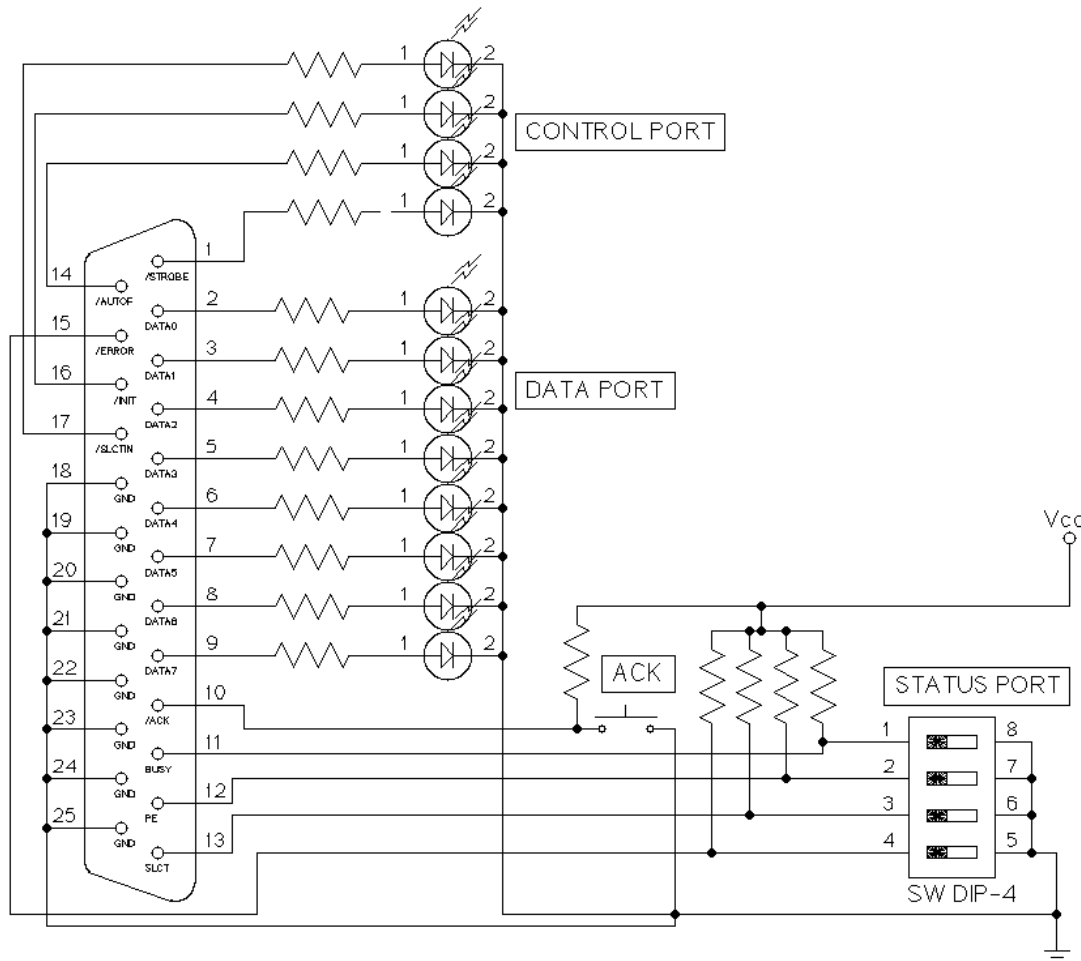


Fig1. Hardware device schematic. The schematic shows the use of simple and low cost components. It is composed by one eight segment display, one interruptor, one switch and a few collection of resistors.

3. Theoretical session

In the first hour of the session, the basic concepts needed for the learning of this activity were exposed. Previous knowledge in the matter of programming in C language and of GNU/Linux operating system were essential to understand the final part of the session.

All the operating systems can be divided in two sections, the first one is related with the user (user space) and a protected one to access to the kernel. In the user space the typical tasks of the normal

activity of a user are done. Nevertheless for managing the devices (configuration, administration, communication...) we needed to work with the superuser privileges. This way of work is called kernel mode [3], whereas first one is called user mode. Usually all the drivers are implemented to work in kernel mode, just because is the only way to have access to the different resources of the operating system.

In most cases, all the operating systems use this implementation, without taking into account the GNU/Linux systems. These systems introduce the kernel module concept. GNU/Linux has two different ways of linking drivers to the system: integrating it in kernel or as a module [4]. The integration of the driver in kernel can only be done if we are completely sure that the driver will work correctly for our device. The introduction of the driver as a module offers advantages to the developer layer since it gives us the capacity to prove each modification of the code, if our device works correctly, so later on we can introduce in the system as a kernel implementation. Modules are frequently used to manage external devices (USBs, HD-USB...), which are not always connected to the system. This is an effective way of saving much memory of the system because it's not necessary to have always in memory all the drivers.

At the end of the theoretical session the process to compile driver was explained. This process consists on the execution of a makefile file that specifies the necessary steps for the installation of the driver in the kernel of the operating system and the verification of its correct operation.

4. Practical session

The second part of the session was the practical one. 2 hours of practice in which the students had to use the concepts of the first part of the session and being aided by the professors, implement a driver for our parallel port device. This practice was made in a classroom with computers by groups of two persons. The idea of this session was to provide to the student the materials for the implementation of the driver and to create their first driver as an example of the methodology followed in the theoretical session. At the end of this practical session, the student must have completed the driver and do it work with the device correctly.

The guideline of the laboratory session was structured in parts so the implementation of the driver was an incremental task. The main process comes from a general GNU/Linux code driver structure and the main functions (reading, writing and the response to the interruptions) had been individually developed until the completion of the driver with the basic functions needed to the proper operation mode and its proper implementation and integration with the operating system.

The incremental implementation of the driver allows to test, on each step of the coding process [5], the new functionality added to the system, and even more important allows to test the learning methodology of the students, with different knowledge levels (not necessarily computing knowledge), were able to understand the implementation of the driver and its operation mode.

The first step in the creation of the driver is to assign it a system identifier (ID) to allow the kernel the identification of the device and to interact and communicate it with the system. This ID is called "Major" [6]. The assignment of this ID is done by the system as an automatic task using the proper function defined in the code. There is another ID called "Minor" that is used to identify other devices needed to the correct operation controlled by the driver.

In GNU/Linux systems the communications with the devices is different than in others operating systems due to the existence of a device manager called "udev" [7], that is in charge of giving support to the rest of devices that are in the user space mode. These devices are in the directory /dev in GNU/Linux systems, which is a special directory to communicate with the peripherals of the system. In this part of

the laboratory session, a node will be created using the proper functions; in other words, the students will create a file in /dev to manage the device and interact with the system.

Following with the implementation process, the basic interacting operations with the device will be created, such as “open”, “release”, “read” and “write”. The functions “open” and “release” are used

when the device is loaded and unloaded from the main memory. “Read” and “write” functions are used as methods for interact with the device. So these functions will describe the operation with the user.

The last part of the creation of the driver, once the node is created and the basics operation implemented, is to communicate through our Input/Output ports [8]. The GNU/Linux kernel offers a simple API (Application Programming Interface) to configure the parallel port and the data transfer. This may not be the most efficient way of working with this port but it is a simple, educative and intuitive way.

Once the node, the main operations and the communication interface are defined, we are ready to compile our new GNU/Linux driver and to integrate it with the operating system. The compilation of the code of the driver was made by the OpenSource Project GCC[9] used by most of the GNU/Linux systems.

After the compilation of the driver, all students tested the device using the basic operations (load, unload, read and write).

5. Results

The results obtained during this activity were succesfull, having into account that in the same session were people of a large variety of knowledges, from primary teachers to computer engineerings and most of them didn't have any previous knowledge of administration or configuration of this operating system. In this activity were 54 people registered and as practical session was not compulsory, only 49 students presented themselves at.the practical session.

The results were: 18 of the 49 students test its final driver working in all the cases using the 2 hours of the session; 3 students finish the driver but this didn't work fine; 13 students didn't finish the code of the driver but there were codifying the lasts functions and the rest couldn't finish the exercise because of its little computers knowledge.

6. Conclusions

These high rates of success considering the large variety of students with from others studies, thus short time to implement the exercise, imply a high effectiveness of the methodology taken on.

The high attendance to the practical session (49 of 54 students) implies that the exposition of the problem was attractive and interesting for the students.

This hardware could be one of the most simple and low cost devices that allow us to teach the concepts and methods needed to develop device drivers in GNU/Linux to a big number of students at the same time.

Acknowledgements The support given by Guillermo González de Rivera, IBM and the UAM.

References

- [1] GNU/ Linux, <http://www.gnu.org>
- [2] Parallel Port, <http://www.beyondlogic.org/spp/parallel.htm>
- [3] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, Third Edition, 942 (2005)
- [4] Linux Modules, <http://tldp.org/HOWTO/Module-HOWTO>
- [5] Drivers en Linux, <http://es.tldp.org/Presentaciones/200103hispalinux/calbet/html/t1.html>
- [6] Alessandro Rubini, Jonathan Corbet, Greg Kroah-Hartman, Linux Device Drivers, Third Edition, 636 (2005)
- [7] udev, <http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html>
- [8] Linux I/O Port Programming, <http://tldp.org/HOWTO/IO-Port-Programming.html>
- [9] GCC, the GNU Compiler, <http://gcc.gnu.org>