Beagleboard Linux Kernel Module

Abraham Arias
e-mail:ipseabraham@gmail.com
Lenin Torres
e-mail:ttvleninn@gmail.com

***Abstract*—Linux kernel modules provide a powerful approach to build a loadable module to the kernel with the power that implies. This project demonstrates an implementation of a simple kernel written in C, which takes advantage of the scope at kernel level, to handle a simple circuit, which anyone can handle by pressing a physical button. At the end there must exists an interaction among the hardware, our software and the kernel itself.**

I. Introduction

The linux kernel uses dynamically loadable modules to simplify and shorten development time, to make configuration easier, and to save kernel memory. This gives a level of flexibility not present in many other Unixes [1]. As part of this project a Loadable Kernel Module (LKM) for Linux kernel 3.2.0 (compatible with 3.x/4.x) is going to be written to interact with the hardware in an embedded device (beagleboard xM Rev C1/C2).

A. Linux Loadable kernel modules

1) Description: On a monolithic kernel (a big single piece of code) when an update is required, all the functions must be relinked, reinstalled and then reboot the system, this is not particularly easy nor fast. Linux developers decided to adopt the microkernel approach, where many kernel functions are separate user-space components and communicate via microkernel facilities, this allows to load and remove modules when it is required. One characteristic of the linux kernel is that it can stack modules by following dependencies, this permits that one module can use other services from another one. This modules are ideal for device drivers, enabling the kernel to communicate with the hardware without it having to know how the hardware works.

Without this modular capability, the Linux kernel would be very large, as it would have to support every driver that would ever be needed on the system. You would also have to rebuild the kernel every time you wanted to add new hardware or update a device driver. The downside of LKMs is that driver files have to be maintained for each device. LKMs are loaded at run time, but they do not execute in user space — they are essentially part of the kernel." [2]
Important aspects are:

a) Their code do not execute sequentially.:
b) Their code do not execute sequentially.:
c) Do not release automatically resources.:
d) They can only 'print' information to the kernel log.:
e) A good module is always on a consistent and valid state, because it can be interrupted at any time.:
f) They can not handle easy floating-point values.:

2) How to start and main functions: A simple module can be build with make on the following Makefile . All for this project can be found online [3].

```
obj-m+=file.o

all:
	make -C /lib/modules/$(shell uname -r)/bui
clean:
	make -C /lib/modules/$(shell uname -r)/bui
```

After building the module (this requires the at least kernel 3, with headers updated,)To load a module a run the following as system administrator, also here are more useful commands [4]:

```
insmod <name>          //to insert a module
insmod <name> name=Nombre //Optional custom parame
lsmod                  //see all the modules at the mo
modinfo <name>         //more information to show
depmod                 //Determine interdependencies
kerneld                //kerneld daemon program
ksyms                  //Display symbols exported for
modprobe               //insert or remove an LKM or se
                       //intelligently
```

The next .c file shows an example of the basic file for a module.

```
#include <linux/init.h>
// Macros used to mark up functions
#include <linux/module.h>
// Core header for loading LKMs into the kernel
#include <linux/kernel.h>
// Contains types, macros,
//functions for the kernel

MODULE_LICENSE("GPL");
//<The license type, this affects runtime behavior
MODULE_AUTHOR("X");
//< The author -- visible when you use modinfo
MODULE_DESCRIPTION("Driver");
MODULE_VERSION("1");

static char *name = "world";
module_param(name, charp, S_IRUGO);
```

```
MODULE_PARM_DESC(name, "/var/log/kern.log");

static int __init helloBBB_init(void){
    printk(KERN_INFO "EBB: Hi %s\n", name);
    return 0;
}
static void __exit helloBBB_exit(void){
    printk(KERN_INFO "EBB: bye %s\n", name);
}
module_init(helloBBB_init);
module_exit(helloBBB_exit);
```

The static keyword is used to restrict the visibility of the function to within this C file. The __exit macro notifies that if this code is used for a built-in driver (not a LKM) that this function is not required.

3) how does a module and a device work?: Starting with insmod, it makes an init_module system call to load the LKM into kernel memory. The init_module call invokes the LKM's initialization routine after it loads the LKM. insmod passes to init_module the address of the subroutine in the LKM name init_module as its initialization routine. Every LKM has a init_module, the same routine exits in the base kernel as well, accessible via subroutine in the standard C library. The LKM author set up init_module to call a kernel function that registers the subroutines that the LKM contains. The LKM's init_module is an ordinary subroutine bound into the base kernel, calling it means branching to its address, the insmod does a query_module system call to find out the addresses of various symbols that existing kernel exports. To see the kind of information insmod can get from a query_module system call, look at the contents of /proc/ksyms. A character device typically transfers data to and from a user application, they behave like pipes or serial ports, like video, audio, among others. The main alternative is a block device, that behave in a similar fashion to regular files, allowing buffered array of cached data to be viewed or manipulated with operations such as reads, writes or seeks. They have major numbers, used by the kernel to identify the correct device driver when the device is accessed and minor number which role is dependent and is handled internally within the driver. The character devices are identified by a 'c' and block devices with a 'b' when the command ls is used. More information about this can be found in the section 3.Data Structures, functions and libraries.

4) Automatic LKM loading and Unloading: If the kernel needs a module, it could be loaded automatically. There are two ways, by kerneld daemon or with the kernel module loader. To understand the second option, let's assume someone is running a program, that executes an open system call for a file in an MS-DOS filesystem, but there are no drivers for it, but identifies that one module previously identified as automatic can be loaded for supply the service.

5) The problem with kernel versions: Since it is possible to compile the kernel and the modules separately, it is also possible to compile them from different source trees. Suppose a module calls a kernel function whose prototype has changed with newer versions of the kernel. Combining the two mismatched codes could cause a system crash. One way to avoid this kind of problem is to store in the module the version of the kernel headers used to compile it to check it when insmod is executed.

Other solution is to perform a 32bit CRC (Cycling Redundancy Check) on each variable, function prototype, and data structure. Symbol names are then mangled with the hexadecimal representation of the CRC giving, for example, jiffies_R2f7c7437or printk_Rad1148ba. When inserting modules, insmod compares the symbols' CRCs. If they match, the variable definition/interface has not changed, and the module may be safely inserted in the usual way. Both the kernel and the module must be compiled with version information for this solution to be effective.

In the future, modules might be extended to cope with even more parts of the kernel, such as memory management. This will take place sooner or later thanks to the free-software spirit and open attitude.

B. Beagleboard and its Operative System

A non profit corporation existing to provide education of the design and use of open-source software and hardware in embedded computing [5]. They provide a low-cost board based on lowpower Texas Instruments processors featuring the ARM Cortex-A series core with all the expandability of today's desktop machines. All this in development environments familiar to just about every developer, from Ubuntu, QNX, Windows Embedded, Android and web tools to bare metal and even Arduino/Wiring-style programming.

1) Introduction to the Beagleboard: Write down what can a Beagle board do is quite difficult, because its flexibility it can work through many environments. Just requires the input voltage and then it can compute almost anything like a desktop computer within its hardware limitations, that is exactly its bottleneck. However it provides a good platform to develop medium projects. All this project is running on the Beagleboard xM Rev1/Rev2. As this board is quite old (2010), some problems on threads have been abandoned, still there are documentation for some problems. The board itself provides a lot of O/I pins, also many peripheral devices can be plugged in, thanks to its USB, HDMI, stereo I/O, USB OTG, connectors and a SD/MMC card slot. This embedded board with 1GHz ARM Cortex-A8 core allows many computations but lacks power to display a desktop of an OS, for this is perfect for system that require such configuration. Like science or engineering projects, without huge amounts of data due to its limitation of course [6].

2) Beagleboard' OS: This board has been demonstrated using Android, Angstrom Linux, Fedora, Ubuntu, Gentoo, Arch Linux ARM and Maemo Linux distributions, FreeBSD, the Windows CE operating system, and RISC OS. With good documentation today we have Angstrom, Debian and Ubuntu, however the first one today does not have support. For this reason the projects runs with Ubuntu 12.04, which has a lot more of documentation and packages than the rest.

C. Specification of the circuit

The first step consist into translate an On/Off signal of 1.8V/0V to some higher voltage on which somes LEDs can

work properly, as shown in figure 1.3.1. Using a 2N222 bipolar transistor as a switch and resistances to create voltage divisors.

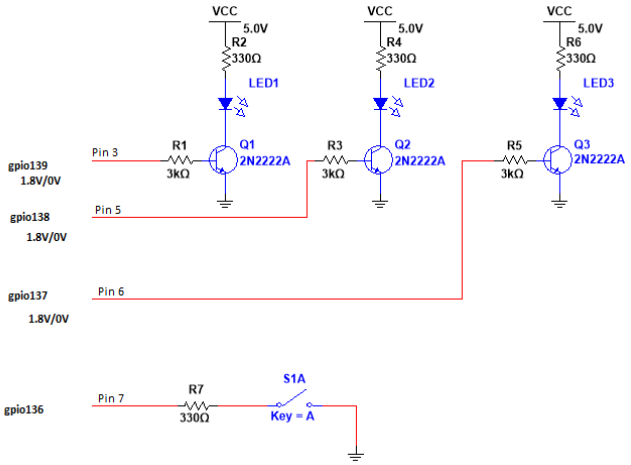

Figure 1. Control Circtuit

Each gpio state, whether is on(1.8V) or off(1.8V) is controlled directly via beagleboard through kernel.

II. Development environment

Here is all the setup to obtain a BeagleBoard on the required state to run this project.

A. Hardware Required

1) Beagleboard xM Rev C1 or Rev C2 [7]
2) Electric DC Adapter of 5V and 2A.
3) MicroSD 4GB with adapter.
4) USB to Serial Cable
5) desktop config(Monitor, keyboard, mouse, ethernet cable).
6) Multimeter and cables.
7) 3 LEDs.
8) 1 Button.
9) Electric jumpers.

B. Software Requiered on the Beagleboard

It is recommend to install the software in following the order on the beagleboard.

1) Ubuntu LTS 12.04 for Texas Instruments OMAP3(Hard-Float)preinstalled desktop Kernel 3.2.0-23-OMAP [8]
2) git
3) gcc compiler

III. Data Structures, functions and libraries

1) module.h : library used to start and stop the LKM, when the user use insmod the program call $module\_init()$ to run the module, end $module\_exit()$ to get out the module from the Kernel.
2) kernel.h: Provide access to kernel virtual memory of linux. Used to create and access to sysfiles in the kernel.
3) interrupt.h: Provide a flags to control the flow of the program. The threads can be interrupted by a irq flags, used in BBLKM to interrupt when the button was pressed.
4) kobject.h: kobject is a struct in the program and have a representation in the sysfs whit files that represent the attributes in the kobject. The files in the sysfs can be modify by the user and this attributes change in the program. When the modules is removed the kobject delete all the directories and files.
5) gpio.h: Library to manipulate the gpio, provide functions to set pin linked to gpio in out or in mode, set the value to low or high and reserve, create and clean a directory with all files that the user need to interact with the pins.
6) time.h: used to get the time in the program.
7) kthread.h: library to use kernel treads and execute functions in parallelisms, used to control the leds burst and to get the button push at the same time.
8) delay.h: library used to sleep the process for a moment to allow time to the pins to change the state.

IV. Instructions to use the program

After satisfying all the requirements from the previous section, that includes your image on the SD. Plug the micro SD, mouse, keyboard, Ethernet and HDMI display, then the power supply, wait several minutes until the initial boot finish, do the configuration setup as in a desktop computer. This is only required in this desktop version, other versions from the first boot you will get everything working from terminal. After this it is possible to use a serial or ssh remote communication. Make sure to have installed gcc, if you want to use the code created in this project install git to clone the code [3].

```
sudo apt-get install gcc
sudo apt-get install git
git clone https://github.com/Abrahamon/BBLKM.git
```

V. Student activity log

Activities references [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27]

VI. Project final status

This project can be used on any Linux system capable of handling output and input gpio, like the Beagleboard, also it needs to have 3.8 kernel or older ones, inside the 3 version, because newer versions use devices tress, and older ones have a different implementation of the module.
In burst mode, if the button is pressed, a counter is increased by one, after executing the burst if exits another on the stack is executed. The communication is not by serial, it occurred by ssh, so to execute the commands the board must be connected to a network.

Table I
ABRAHAM'S ACTIVITIES

Date	Hours	Activity
30/04	3	$Downloadseveral new O.S. for the BB.$ Have to find an appropriate voltage cable, one USB is not working, due to fail to boot.
03/05	4	Meeting. Update packages, connect through ssh. Use git with our project. Install headers.
04/05	5	Meeting. We tried to turn on or off the gpio pins, they do not seem to work at all, some suggest it is a matter of the pin mux. Obtain another BB from electronic school to double the work.
05/05	10	Meeting. Tried to use the Mux for the pins, now I use the serial cable. Many attempts more to use the gpio on Ubuntu, Angstrom, Debian on differents versions. The pins only work on the latest of Angstrom. In 3.8 kernel or newer this typical method does not work. Gcc in angstrom but found problems to build the module.
06/05	4	Read files from the LKM, Install all the required software to handle pins, pins work, but can't build the module. This is the holy version that handle pins and has kernel 3, Start with the documentation.
10/05	2	Documentation
11/05	4	Finished the docu
Total	42	

Table II
LENIN'S ACTIVITIES

Date	Hours	Activity
30/04	4	Research for linux kernel and how to make a module
01/05	3	Read the examples provided by the professor and write first helloworld.c and Makefile to compile and create the hello.ko and add this to kernel with insmod.
03/05	12	Investigate how to work with the pins of the Beagle Board with some different operative system.
07/05	4	Partner found how use the pins in ubuntu and I prove a new module with gpio interact and messure the voltage in the pins to work with leds and button
09/05	13	Write the program to interact with the kernel with our circuit and make test with different files.
11/05	3	Documentation
Total	39	

VII. CONCLUSIONS, SUGGESTIONS/RECOMMENDATIONS

A. Linux kernel modules provide a powerful interface to control hardware when is required, and without the need to compile our kernel again. This modules running at kernel level allow users to use its services and also other program at the level of kernel can use its services.

B. Many new O.S. presented problems handling I/O pins of the beagleboard, due to modification to the device tree after kernel 3.18. In the BeagleBone apparently this problem is not present and the management of the gpio port comes naturally. The old OS are not useful due to lack of support to their packages, also they have older versions of kernel like 2.X. Besides the beagleboard requires OMAP3, and operative systems nowadays support OMAP4. The recommendation here is to use Ubuntu 12.10, which handles OMAP3 and has the kernel 3.2, which is recommended to develop LKM.

C. The malicious loadable kernel module is already publicly available and probably used in many intrusions. It raises the bar of compromise and incident response to the kernel level. Although it may seem that kernel modules are a significant factor when performing investigations, they are not lethal. You can see that there are simple preventative measures that you can take to protect yourself against this type of compromise. Furthermore, investigative tools and techniques for this type of compromise fight fire with fire by also executing in kernel space.

retrieved May 11, 2016.

[2] D. Molloy, "Writing a linux kernel module," http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/, 2015, retrieved May 11, 2016.

[3] V. Arias A., Torres, "Bblkm," https://github.com/Abrahamon/BBLKM, 2016, retrieved May 11, 2016.

[4] B. Henderson, "Linux loadable kernel module howto," http://www.cs.uccs.edu/ cchow/pub/linux/kernel/Module-HOWTO.pdf, 2002, retrieved May 11, 2016.

[5] B. Fundation, "About beagleboard.org," http://beagleboard.org/about, 2016, retrieved May 11, 2016.

[6] D. Molloy, "Exploring beaglebone: Tools and techniques for building with embedded linux," 2014.

[7] B. Fundation, "Beagleboard xm system -reference manual," http://beagleboard.org/static/BBxMSRM_latest.pdf, 2010, retrieved May 11, 2016.

[8] Ubuntu, "Ubuntu 12.04.5 lts (precise pangolin)," http://cdimage.ubuntu.com/releases/12.04/release/, 2014, retrieved May 11, 2016.

[9] K. Kanetkar, "Port ubuntu on the beagleboard," http://ecee.colorado.edu/ecen5623/ecen/labs/Linux/Tutorials/Ubuntu_on_beagle.pdf, 2014, retrieved May 11, 2016.

[10] "Beaglebone black doesn't sometimes start. only power led is on," https://groups.google.com/forum/#!topic/beagleboard/aXv6An1xfqI%5B1-25%5D, 2015, retrieved May 11, 2016.

[11] "Beaglebone black doesn't sometimes start. only power led is on," http://electronics.stackexchange.com/questions/84699/how-much-power-can-be-supplied-over-beaglebone-blacks-usb, 2015, retrieved May 11, 2016.

[12] "Recommended beagleboard peripherals," http://beagleboard.org/peripheral, 2014, retrieved May 11, 2016.

[13] M. works, "Configure network connection with beagleboard hardware," http://www.mathworks.com/help/supportpkg/beagleboard/ug/getting-thebeagleboardipaddress.html?s_tid=gn_loc_drop, retrieved May 11, 2016.

[14] V. Gite, "Debian linux install openssh sshd server," http://www.cyberciti.biz/faq/debian-linux-install-openssh-sshd-server/, 2012, retrieved May 11, 2016.

[15] "Using gpios on beagleboard -xm," https://tekytech.wordpress.com/topics/using-gpios-on-beagleboard-xm/, retrieved May 11, 2016.

[16] "Pin mux tool," http://www.ti.com/tool/PINMUXTOOL#descriptionArea, retrieved May 11, 2016.

[17] "Beagleboardpinmux," http://elinux.org/BeagleBoardPinMux, retrieved May 11, 2016.

[18] Rohit, "[beagleboard-xm] pin muxing and gpio tutorial," http://technologyrealm.blogspot.com/2014/02/beagleboard-xm-pin-muxing-and-gpio.html , 2013, retrieved May 11, 2016.

[19] ash w., "Tutorial: Using the dsp on the beagleboard," http://ash-wilding.blogspot.in/2013/08/tutorial-using-dsp-on-beagleboard.html, 2014, retrieved May 11, 2016.

[20] "Bootloader fails to load linux kernel," http://unix.stackexchange.com/questions/122582/bootloader-fails-to-load-linux-kernel, 2014, retrieved May 11, 2016.

[21] "Installing gcc," https://gcc.gnu.org/wiki/InstallingGCC, 2016, retrieved May 11, 2016.

[22] "Installation of gcc in angstrom on beagleboard-xm : All technical support docs will be available on git," http://blogspot.tenettech.com/?p=1370, 2012, retrieved May 11, 2016.

[23] "Reading files from the linux kernel space," https://www.howtoforge.com/reading-files-from-the-linux-kernel-space-module-driver-fedora-14, retrieved May 11, 2016.

[24] "Helper scripts for setting an angstrom development environment," https://github.com/angstrom-distribution/setup-scripts, 2015, retrieved May 11, 2016.

[25] O. P. Peter Jay Salzman, "The linux kernel module programming guide," http://www.tldp.org/LDP/lkmpg/2.4/lkmpg.pdf, 2003, retrieved May 11, 2016.

[26] "Remote serial console howto," http://tldp.org/HOWTO/Remote-Serial-Console-HOWTO/upload-kermit.html, retrieved May 11, 2016.

[27] D. Molloy, "Building git from source for the beaglebone black," http://derekmolloy.ie/building-git-beaglebone/, 2013, retrieved May 11, 2016.