Alessandro Osima
Milan, 22 April 2013

Evdev like userspace/kernelspace communication system for input devices

About you

What is your name?
Alessandro Osima

What is your email address?
alex.osima@gmail.com
alex_osima@yahoo.it

What is your eLinux.org wiki username?
Asion

What is your IRC nickname?
asion

What is the name of your School and in what country?
Italy, Università Degli Studi di Milano pursuing a B.Sc. in computer science

What is your primary language?
Italian

Where are you located, and what hours do you tend to work?
I live in Milan (GMT +2). During summer I usually work from 11:00 a.m to 4:00 p.m and/or from 11:00 p.m to 3:00 a.m but I can easily change my work schedule to adapt to my mentor's needs

Have you participated in an open-source project before?
I have never directly contributed to an open-source project but I have used a lot of open source products like the raspberry pi and various linux distros both personally and for my university studies.

I think I have enough experience to give something back to the open-source community and to help make better software that everyone can freely use and improve.

About your project

What is the name of your project?
Evdev like userspace/kernelspace communication system for input devices

Describe your project in 10-20 sentences. What are you making? For whom are you making it, and why do they need it? What technologies will you be using?
This project aims to build a new kernel subsystem designed to offer an api for input device drivers and input subsystems to communicate with userspace throughout file operations on files residing in the /dev directory.
For every device driver that will register with nevdev (new evdev) a new file will be placed in the /dev/nevdev directory, ready to intercept read/write/ioctls syscalls from userspace applications.
To remain abstract from a particular subsystem nevdev will not handle these calls directly or implement some kind of global interface. It will just redirect them to the proper device, granting every subsystem, like iio or input, the possibility to offer its own interface that can then be recovered by userspace trough ioctls calls.

This project was initially focused on offering an iio drivers debug interface but thanks to some advice of my would-be mentor Hunyue Yau, I turned to build a more robust and general interface based on the evdev design.
This project will provide driver developers, who use iio or input subsystems, a common way to communicate with userspace applications resolving a problem that is troubling iio developers, as explained in this post http://us.generation-nt.com/answer/rfc-iio-options-event-userspace-interface-help-203768792.html.

Evdev itself can not be directly used, because it is too entangled with the input subsystem to be efficiently abstracted from it. Anyway a part of the code of evdev and its basic approach are reused in nevdev.
Also, Nevdev design will ensure backward compatibility with the current evdev userspace event-based api, thus requiring no changes in userspace applications (like evtest) or libraries.

Nevdev will offer a simple api based on the register/unregister functions like other kernel subsystems.

Internally nevdev will be composed by a list of handles corresponding to all the registered devices.
When a new device is registered, nevdev will create a new handle containing a struct device, a cdev and a file_operations structure.
The registering driver will provide its device structure to be used by nevdev as the new handle's device parent and a file_operations structure.
The hotplug event resulting from the handle device structure allocation and registration will be used by a simple udev script to populate the /dev/ nevdev directory.
All the the file i/o calls executed on /dev/nevdev on the newly registered file will be then intercepted and sent to the registered driver file_operations structure thanks to the allocation of the handle cdev structure with the handle's device struct major and minor numbers.

What is the timeline for development of your project? The Summer of Code work period is about 11 weeks long; tell us what you will be working on each week.

Period of time	Goals
May	In my freetime I will gain more experience with the beaglebone, its modded 3.x kernel, the input and iio subsystems by writing a couple of test drivers, doing some evdev modding and reading more kernel source code. I will also keep in touch with my mentor so that I can start coding as soon as gsoc begins.
Community bonding period	Review my project and my work schedule with my mentor and fix eventual shortcomings.
17 june - 7 july	Implement the basics of nevdev: 1) subsystem initialization and de-initialization 2) device adding and removal to nevdev 3) implement all the concurrency primitives
7 july -14 july	Test the code
14 july - 4 august	1) write the final kernel side api 2) write the udev script to export the sys files in a /dev subfolder and write a cdev
4 august - 11 august	Test the code
11 august - 25 august	1) integrate cdev into the codebase to receive file i/o callbacks from /dev 2) write a subsystem wide ioctls interface

Period of time	Goals
25 august - 1 september	Test the code and start writing some documentation
1 september - 8 september	Implement file_operations redirection to the registered subsystems trough their file_ops structures obtained during device registration
8 september - 23 september	1)test the final code 2)write documentation and examples 3)submit the code for final evaluation

Convince us, in 5-15 sentences, that you will be able to successfully complete your project in the timeline you have described.

I have been working with C/C++ for over 4 years. I started studying it by myself because I was curios about game programming and after a while I became more interested to the inner working of operating systems rather than games.

For this reason I started using linux and reading its source code.

I kept following this passion when I decided to study computer science.

While at university I learned a lot more about operating systems and computer architecture.

Six months ago I bought a raspberry pi and started working with it.

Thanks to the pi I refined my knowledge of electronics and low level programming.

Over the years I have written some simple games using opengl and c++.

For my computer architecture exam I created a cuda application designed to efficiently apply a median filter on a matrix of pixels. On the pi I wrote various programs to take advantage of the gpio and spi interfaces, I have also done some programming with code running directly on the board without using an operating system. You can find some of this projects on my bitbucket page https://bitbucket.org/dashboard/overview.

During the coding weeks:

I plan to keep a github repository with all the code and documentation I will produce.

Every week I will post on the mailing list and on irc a progress report documenting all the tasks I have completed and the ones I will do the following week.

Also I will coordinate with the linux-input community and request comments, opinions and reviews to increase the chances of an inclusion in the mainline kernel.

Goals:
The project will be initially based on the current beaglebone kernel and use the bone as the main testing platform.
I will not use any beaglebone specific part of the kernel because I aim to upload it in the mainline linux kernel. This will probably happen at the end of the project or even later on, because of the long time possibly required for a successful inclusion upstream.

You and the community

If your project is successfully completed, what will its impact be on the BeagleBoard.org community?
This project would help not only the beagleboard community but the entire linux-input drivers developers community by offering a simple clean interface that can be used by most input device drivers to communicate easily with userspace.
With nevdev everything a driver has to do is to register his device structure and intercept the file i/o calls, letting the new subsystem do all the rest of the job.

One problem that this project can easily solve is the one detailed in this year project ideas page by Hunyue Yau related to iio debugging drivers. IIO can set up an userspace interface by just registering a device with nevdev and then handle all the read file_operations on that device.
Then a userspace program can receive all the events sent from an iio device with a simple read on the corresponding device file in the /dev directory.
This approach can be easily replicated with most input device drivers.

What will you do if you get stuck on your project and your mentor isn't around?
First thing I will debug the code over and over to be sure I'm really stuck and not only momentarily lost.

If unsuccessful I will think about different ways to resolve my problem helping myself with an exhaustive search on books, google, forums, mailing list and irc backlogs.
In the very unlikely case all the previous steps still left the problem unresolved I will probably ask on IRC or on a mailing list if someone has encountered a similar problem before.