How and why to work with the Kernel Community

FreedomHEC Taipei 2008

by

Harald Welte <HaraldWelte@viatech.com>

Linux Developer
Consultant
VIA Open Source Liaison

Introduction

Who is speaking to you?

- an independent Free Software developer, consultant and trainer
- 13 years experience using/deploying and developing for Linux on server and workstation
- 10 years professional experience doing Linux system + kernel level development
- strong focus on network security and embedded
- expert in Free and Open Source Software (FOSS) copyright and licensing
- digital board-level hardware design, esp. embedded systems
- active developer and contributor to many FOSS projects
- thus, a techie, who will therefore not have fancy animated slides ;)

Introduction

What is my affiliation with VIA

- First contact with VIA in Q3/2007
- I became VIA Open Source Liaison in July 2008
- In this role, I help the VIA Linux Committee, PM's and Engineers
  - to understand the Linux and Open Source world
  - to communicate with the Open Source world
  - to interface to Linux community concerns and take them to VIA
  - to get VIA on track for world-class Linux support
- I percieve myself more as Linux person inside VIA, not VIA person in Linux ;)
- Sorry: I am an expert on Linux, not [yet] on VIA's products
  - so please excuse me if I say something wrong about VIA hardware
- I don't speak for VIA, just for myself

What is Free Software?

- Software that is
  - available in source code
  - is licensed in a way to allow unlimited distribution
  - allows modifications, and distribution of modifications
  - is not freeware, but copyrighted work
  - subject to license conditions, like any proprietary software
  - READ THE LICENSE

What is Open Source?

- Practically speaking, not much difference
- Remainder of this presentation will use the term FOSS (Free and Open Source Software)

What is the FOSS Community?

- Diverse
  - any individual can contribute
  - no formal membership required
  - every project has it's own culture, rules, ...
- International
  - the internet boasted FOSS development
  - very common to have developers from all continents closely working together
- Evolutionary
  - developers come and go, as their time permits
  - projects evolve over time, based on individual contributions

Development Process

- "Rough concensus and running code"
- Decisions made by technically most skilled people
- Reputation based hierarchy
- Direct Communication between developers
- Not driven by size of a target market
- Release early, release often

FOSS Community likes

- generic solutions
- portable code
- vendor-independent architecture
- clean code (coding style!)
- open standards
- good technical documentation

FOSS Community dislikes

- monopolistic structures
  - e.g. intel-centrism
- closed 'industry forums' with rediculous fees
  - e.g. Infiniband, SD Card Association
- standard documents that cost rediculous fees
- NDA's, if they prevent development of FOSS

The "Linux" System

- What is a so-called Linux system
  - The Linux operating system kernel
  - The X.org X11 windowing system
  - Various non-graphical system-level software
  - A variety of different desktop systems (KDE, Gnome)
  - A variety of GUI programs
- In reality, this is a "Linux Distribution"
  - sometimes referred to as "GNU/Linux System"

Entities in the Linux system

- Free Software projects and their developers
- So-called "Distributors" who create "Distributions"
- Contributors
- Users
- Vendors of proprietary Linux software

FOSS Projects

- Free Software projects and their developers
  - Linux Kernel, Xorg, KDE, Gnome, Apache, Samba
- Role
  - Development of the individual program
  - Very focused on their individual project
  - Portability and flexibility usually main concern
  - Interact based on practical neccessity
  - Usually they just provide source code, no object code

Distributions

- Distributions (both commercial and community based)
  - Debian, Ubuntu, SuSE, Fedora, RedHat, Mandriva, ...
- Role
  - Aggregate thousands of individual FOSS programs
  - Find stable and compatible versions of those programs
  - Do 'software system integration'
  - Offer bianary software packages and installation media
  - Offer (security) updates to their users
  - Offer free/best effort or commercial support for professional users

Contributors

- Contributors
  - are people not part of a specific development team
  - usually "very active users" of a particular program
- Role
  - find / document / fix bugs that they find themselves
  - contribute bug reports, documentation or code
  - participate in discussion on features or problems

Collaborative Software Development

- How do projects communicate internally
  - Very rarely in physical meetings (people live too far apart)
  - Very rarely in phone conferences (people live in different timezones)
  - It's almost entirely text-based (e-mails, sometimes chat system)
- Mailing Lists
  - Usually every project has at least one list
  - Often there are separate lists for developers and users
  - Participation in the mailing list (reading and posting) open to anyone

Collaborative Software Development

- Project Management / Decision making
  - usually there's a small group (coreteam) or one leader
  - he is often the creator of the program, or it's maintainer
  - he has the final say in what is accepted or not
  - larger projects have 'subsystem maintainers' with delegated authority
  - so quite often, the structure is more hierarchical than people believe
  - rough concensus and running code

Linux and binary compatibility

- Linux and binary compatibility
  - Drivers usually run inside the OS kernel
  - Linux doesn't have any stable kernel-internal ABI
  - Linux doesn't even have stable kernel-internal API
  - Only the ABI to userspace is stable/fixed
- Thus, every minor Linux release can break in-kernel ABI+API
- This is why binary-only drivers simply don't work!

Linux and binary compatibility

- I still don't believe! Why not binary-only drivers
  - because every distribution has a different base kernel revision
  - because every distribution can change their kernel version e.g. as part of a security update
  - users will end up in incompatibility nightmare
  - so please, don't do it. It will never work for the majority of your users

Implications for Hardware Vendors

- Implications for Hardware Vendors
  - Users are used to get all software from the distribution
  - They are not used to separate vendor-provided driver CD's
  - Thus, drivers need to be in the distribution
  - Goal: getting drivers into the distrubution

Implications for Hardware Vendors

- How to get drivers into distributions?
  - You can talk directly to the distributions
  - But: Their code architecture/style requirements are high
  - But: Many of them do not accept binary-only drivers
  - But: There are many, many distributions.
  - Linux is only a certain portion of the market
    - Every distribution is only a small portion of the portion
    - Thus, new goal: Get your drivers in the mainline project

Implications for Hardware Vendors

- Getting drivers in the mainline project
  - ensures that all distributions will pick up the driver
  - ensures out-of-the box support of your hardware on all distributions
  - ensures best user experience
  - ensures least internal R&D resources
    - no need to provide binaries for 3 versions of 5 distributions
    - no need to constantly try to catch up with distribution kernel updates

Windows driver development model

- MS defines stable APIs and ABIs for drivers and releases SDK (DDK)
- All interfaces are specified by a single entity
- The interface between driver and OS core is designed as binary interface
- Hardware vendors develop drivers for their hardware component
- Hardware vendors compile and package drivers for their hardware component
- Hardware vendors sell bundle of hardware and software driver (object code)

Linux driver development model

- A community-driven process creates in-kernel driver API's
- Drivers are written against those APIs
- Drivers are submitted to the kernel developes for inclusion into the OS source tree
- Because all (good) drivers are inside one singe source tree, OS developers can (and will) refine the APIs whenever apropriate
- There are no stable in-kernel API's, and especially no stable in-kernel ABI's
- Linux development community releases kernel source code
- Hardware vendor sells hardware only. The Windows driver CD is unused.

Linux driver development model

- Without proper support from HW vendor, Most hardware drivers are developed by people inside that community
  - sadly most of them have no relation to the HW manufacturer
  - even more sadly, many of them have to work without or with insufficient documentation (reverse engineering)
- Good HW vendors understand this and support Linux properly!
- Linux is a big market by now
  - Servers
  - Embedded devices (est. > 40% of all wifi/dsl router + NAS appliances)
  - Increasingly popular on the Desktop
  - Recently: Netbooks

Linux driver development model, bad case timeline

- Hardware vendor produces and ships hardware
- Users end up getting that hardware without any Linux support
- Somebody will start a driver and inquire about HW docs
- Hardware vendor doesn't release docs
- If hardware is popular enough, somebody will start reverse engineering and driver deevlopment
- With some luck, the driver is actually useable or even finished before the HW product is EOL

Linux driver development model, good case timeline #1

- Hardware vendor starts Linux driver development for new HW during HW R&D
- Hardware vendor submits Linux driver for review / inclusion into mainline Linux kernel before HW ships
- User installs HW and has immediate support by current Linux kernel
- Hardware vendor publicly releases HW docs when the product ships, or even later
  - This enables the community to support/integrate the driver with new interfaces
  - It also enables the community to support hardware post EOL, at a point where the HW vendor

Linux driver development model, good case timeline #2

- Hardware vendor releases HW documentation during HW R&D or no later than the product start shipping
- Somebody in the Linux development community might be interested in writing a driver
  - in his spare time because of technical interest in the HW
  - as a paid contractor by the HW vendor
- In such cases it helps if the HW vendor provides free samples to trustworthy developers
- That driver is very likely to get merged mainline

Why submit your code mainline?

- Quantity-wise, most users use some Linux distribution
- Every version of every distribution ships a different Linux kernel version
- Most end-users are not capable of compiling their own kernel/drives (but way more than you think!)
- Thus,
  - teaming up with one (or even two, three) Linux distributions only addresses a small segment of the user base
  - distributing your driver independently (bundled with hardware, ...) in a way that is ready-to-use for end-users is a ton of work and almost impossible to get right
  - the preferred option, with the least overhead for both user and HW vendor is to merge the driver mainline.

How to submit your code mainline?

- The FOSS code quality requirements are _extremely_ high
- It's not a surprise that Linux is generally considered much more stable than competitors
- Code needs to be maintainable
  - Linux supports old hardware ages beyond their EOL
  - Thin of MCA, VLB, Decnet, IPX networking, ...
- So unless you respect the development culture, your code is likely to get rejected!
- Post your driver at the respective mailing lists
- Release early, release often
- Don't hesitate to ask for feedback and suggestions if you are not 100% sure what is the right way to implement a certain feature

What about other FOSS OS's

- There are quite a number of other non-Linux FOSS OSs, among them
  - FreeBSD, OpenBSD, NetBSD, ...
- Those are not as small as you might think
  - FreeBSD often used for internet severs (web, mail, ...)
  - OpenBSD often used in high-security environments
  - NetBSD a little more prominent in embedded
- So how does this affect a HW manufacturer
  - In case the OS is used in a targetted market, developing a driver might make sense
  - In most cases, open docuentation is all those projects need
  - In other cases, dual-licensing a driver (GPL+BSD) makes sense so *BSD can use code from the Linux driver

Techncal differences

- In the MS world, almost all interfaces are MS defined
- In the Linux world, Linux is only the OS kernel
- All other interfaces are specified by their respective projects
- Often there are many alternatives, e.g. for graphical drivers
  - X.org project (X11 window server, typical desktop)
  - DirectFB project (popular in embedded devices like TV set-top boxes)
  - Qt/Embedded (popular in certain proprietary Linux-based mobile phones)
- Every project has it's own culture, including but not limited to
  - coding style
  - patch submission guidelines
  - software license
  - communication methods

Practical Rules

- 1. Much more communication
  - It's not a consumer/producer model, but cooperative!
  - Before you start implementation, talk to project maintainers
    - It's likely that someone has tried a similar thing before
    - It's likely that project maintainers have already an idea how to proceed with implementation
    - Avoid later hazzles when you want your code merged upstream

Practical Rules

- 2. Interfaces
  - If there is a standard interface, use it
  - If insufficient: Don't invent new interfaces, try to extend existing ones
  - If there is an existing interface in a later (e.g. development) release upstream, backport that interface
  - Don't be afraid to touch API's if they're inefficient
    - Remember, you have the source and _can_ change them

Practical Rules

- 3. Merge your code upstream
  - Initially you basically have to create a fork
  - Development of upsteram project continues sometimes at high speed
  - If you keep it out of tree for too long time, conflicts arise
  - Submissions might get rejected in the first round
    - Cleanups needed, in coordination with upstream project
    - Code will eventually get merged
  - No further maintainance needed for synchronization between your contribution and the ongoing upstream development
  - Don't be surprised if your code won't be accepted if you didn't discuss it with maintainers upfront and they don't like your implementation

Practical Rules

- 4. Write portable code
  - don't assume you're on 32bit CPU
  - don't assume you're on little endian
  - if you use assembly optimized code, put it in a self-contained module

Practical Rules

- 5. Binary-only software will not be accepted
  - yes, there are corner cases like FCC regulation on softradios
  - but as a general rule of thumb, the community will not consider object code as a solution to any problem

Practical Rules

- 6. Avoid fancy business models
  - If you ship the same hardware with two different drivers (half featured and full-featured), any free software will likely make full features available on that hardware.

Practical Rules

- 7. Show your support for the Community
  - By visibly contributing to the project
    - discussions
    - code
    - equipment
  - By funding developer meetings
  - By making rebated hardware offers to developers
  - By contracting / sponsoring / hiring developers from the community

Thanks

- Please share your questions and doubts now!
- Please contact me at any later point, if you have questions
- I'm here to help understand Linux and Open Source!

- HaraldWelte@viatech.com
- laforge@gnumonks.org
- hwelte@hmw-consulting.de

Thanks for your Attention