

Introduction to the Linux Development Model for Hardware Companies

by

Harald Welte <hwelte@hmw-consulting.de>

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 hardware design, esp. embedded systems
- active developer and contributor to many FOSS projects
- currently lead system architect (hardware + software) for OpenMoko
- thus, a techie, who will therefore not have fancy animated slides ;)

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 its own culture, rules, ...

International

- the internet boosted 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

People / Groups involved

- Really depends on size of projects
- Small projects often a one-man show
- Bigger project have groups / subgroups
- Common Terms / Definitions
 - Maintainer
 - ▷ The person who formally maintains a project
 - Core Team / Steering Committee
 - ▷ A group of skilled developers who make important decisions
 - Subsystem Maintainer
 - ▷ Somebody who is responsible for a particular sub-project
 - Developer Community
 - ▷ All developers involved with a project
 - User Community
 - ▷ Users of the software who often share their experience with others

Development Process

- "Rough consensus 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

Motivations

- gaining reputation (like in the scientific community)
- (students) gaining development experience with real-world software
- solving problems that the author encounters on his computer
- fighting for Free Software as ideology
- working on exciting technology without having to work at company XYZ
- work in creative environment with skilled people and no managers ;)

FOSS Community likes

- generic solutions
- portable code
- vendor-independent architecture
- clean code (coding style!)
- open standards
- good technical documentation
- raw hardware, no bundle of hardware and software sold as solution

FOSS Community dislikes

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

Weak Points of FOSS

When foss is entirely volunteer-driven

- often way behind schedule (if there is any)
- already too late when projects start
 - started when there already is a real need
- often a lack of (good) documentation
 - programmers write code, not enduser docs...
- strong in infrastructure, weak in applications
 - traditionally developers interested in very technical stuff

- Thus, FOSS really improves when organizations/entities get involved the right way!

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 developers for inclusion into the OS source tree
- Because all (good) drivers are inside one single source tree, OS developers can (and will) refine the APIs whenever appropriate
- 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

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 development
- 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/drivers (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
 - ▷ Think 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 servers (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 docuementation 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

Technical 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 upstream 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 soft radios
 - 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

Thanks for your Attention