

Introduction to the Linux Development Model

by

Harald Welte <hwelte@hmww-consulting.de>
[hmww-consulting](http://hmww-consulting.org) / gpl-violations.org / openmoko.org

Introduction

Who is speaking to you?

- an independent Free Software developer, consultant and trainer
- 14 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?
- an independent freelancer, not speaking for any company
 - working in the Free Software community for many years
 - used to be the maintainer of the Linux firewall netfilter/ip tables
 - started many Free Software and Open Hardware projects, e.g.
 - ▶ OpenEZX - Open Source for Motorola EZX phones
 - ▶ OpenPCD/librfid - 13.56MHz RFID stack
 - ▶ OpenBeacon - 2.4GHz active RFID
 - ▶ gnufish - Linux for E-TEN PDA-phones
 - ▶ OpenBSC - A GSM backend network BSC+MSC+HLR
 - was employee #1 and Lead System Architect (HW+SW) of Openmoko
 - consulting many companies on FOSS development + licensing□

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
- **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 always driven by size of a target market
- Release early, release often

Motivations (individual)

- 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 ;)

Motivations (corporate)

- not having to reinvent the wheel
- if FOSS provides 80% of your problem solution, you just have to add the missing 20%
- fully customizable, every aspect of the system can be modified/adopted/changed
- no per-unit royalties
 - be aware, you have more one-time R&D cost

Who is "The Community"?

- Studies show
 - the majority of the Linux kernel code is developed by professional, paid developers
 - most of them work for large IT companies (Intel, Novell, IBM, RedHat, ...)
 - those companies would not invest the development resources if there was no business case for it!
- So "the community"
 - is not a random collection of individuals scratching their itch
 - but is a group of very prominent professional developers working for some of the biggest IT companies worldwide

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
 - note: Samsungs manuals now under NDA :(

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 commercial 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 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 aproprate
- 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 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?

- In the PC world
 - 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.

Why submit your code mainline?

- In the embedded/ARM world
 - there are more customers of your SoC than just the tier-1 custom
 - the small/medium size customers do not qualify for your support
 - ▶ but if documentation and/or source is available, they can still buy and use your product
 - the more developers know your product, the more will recommend it in their companies
 - ▶ existing experience with a certain SoC is very valuable, reduces lead time, helps solving problems quickly
 - there are even way more custom distributions in the embedded world
 - ▶ you can never support even the smallest fraction of them
 - ▶ but all of them use the mainline kernel as base version
 - ▶ if your driver + support code is in mainline, all of the distributions will easily run on your SoCs
 - keeping all code in mainline reduces fragmentation of the codebase

Samsung LSI is part of the community

Samsung LSI is part of the community

- Linux exists because of massive, industry-wide collaboration
- Only because everyone contributes, Linux grows
- Everyone helps to create a better platform
- If SLSI Linux drivers/support is good, Linux customers prefer SLSI over other vendors
- Don't only create drivers, but infrastructure (core OS/kernel)
- Every company does its small part of the Linux

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

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 its 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 software
 - ios
 - 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

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 necessity
- 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 binary 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

Users

- **Users**
 - are people just using software
- **Role**
 - using programs
 - they usually just install+use a particular distribution
 - they typically do not download+install software directly from the particular software project

Vendors of Proprietary Software

- Vendors of proprietary Software (e.g. Oracle)
 - remain a small niche in the Linux world
 - usually driven by a very specific industry
 - they can exist because kernel/userspace ABI is stable!
- Role
 - feed-back some of their requirements to the Open Sourcedevelop
 - help the operating sytstem development to make sure OS is good for them
- Note: This is not applicable for driver developm ent!
 - drivers are in the Linux kernel, not userspace

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 systems)
- 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
 -

Motivation of Software Developers

- Why do developers work on a FOSS project
 - because they're interested in a certain area
 - because it's fun to learn and improve skills
 - because it's fun to co-work with world-class hackers
- How do people make money
 - often by offering commercial support for their software
 - by offering porting or system integration
 - by offering development of extensions/modifications
 - by working for a company that uses/needs that program

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 distribution

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

Thanks

- Please share your questions and doubts now!
- Please contact me at any later point, if you haveques-
- I'm here to help Samsung!
- hwelte@hmw-consulting.de

Thanks for your Attention