Introduction to the Linux Coding Style

by

Harald Welte <hwelte@hmw-consulting.de>

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 syste ms
- active developer and contributor to many FOSS projects
- thus, a techie, who will therefore not have fancy animated slides ;)

Code Architecture / Style

- What is coding style ?
  - It is not just about cosmetics / code format and layout
  - It is a fundamental skill of sustainable software engineering
  - It is about writing readable, not just executable code
  - It is about clearly expressing your thoughts and ideas
  - It is about good software architecture

Code Architecture

- Why does good code architecture matter ?
  - Because Linux runs on 25 CPU architectures
  - Because Linux runs on systems with 1 or 512 CPU cores
  - Because Linux is a reliable operating system kernel
  - Because Linux will support your hardware even after the hardware vendor doesn't
    - becuase the company is gone
    - because the company has lost business interest
    - because the original developers are gone

Code Architecture

- Linux kernel API's change
  - the kernel constantly gets improved
  - the kernel constantly adapts to changes in e.g. hardware
- Use latest kernel API's
  - very often there are old and new API's in parallel
  - old API's are only to be used by legacy drivrers until they have been converted to the new API's
  - new drivers using old API's will not get merged

Code Architecture

- Code reuse
  - makes software maintainable
  - makes software vendor-independent
  - increases performance (efficient memory+cache use)
  - so please, reuse existing code
  - decreases overall R&D effort
  - example
    - Linux provides one 802.11 stack for all wifi cards
    - Linux provides one Bluetooth stack for all bluetoth HCI
    - Vendor drivers only implement minimal hardware glue

Code Architecture

- Code Structure
  - helps code to be readable
  - helps code to be maintainable
- means
  - functions of reasonable length
  - no spaghetti code
  - functions with clearly-defined purpose

Code Architecture

- Code Portability
  - Linux runs on 25 CPU architectures
    - some 32bit/64bit
    - some cache-coherent, some not
    - some with CPU == IO address space, some not
    - some little, some big endian
    - with different alignment requirements
    - with or without SMP
  - So please never, ever assume you only care about IA32.

Coding Style

- Coding style in a narrow sense
  - is how the code actually looks like
  - /usr/src/linux/Documentation/CodingStyle
- Why do "cosmetics" matter
  - you write code to be read by other developers
  - anyone who reads one part of the kernel should be able to read all parts
  -

Coding Style

- Indentation
- No multiple statements on one line
- Break long lines to fit 80character terminal wid th
- Opening/closing braces on same line, except fun ctions
- No unneccessary braces
- Space after keyword, but not after function
- No space inside parenthesis

Coding Style

- Centralized exitting of functions
  - goto helps
- C89 style comments
  - /* */ instead of //
- careful with inlining
  - excessive inlining wastes cache
- function return values
  - standard case: 0 in success, -ERRNO on error
- volatile is almost always wrong
  - see Documentation/volatile-considered-harmful.txt

Coding Style

Naming

- DontUseSturdyCapsLikeInWindows
- keep local variables short
- global symbols with prefix and underscore
  - like s3cfb_do_something()

Coding Style

Now, let's look at some actual code!

Why does revision control matter

- because revision control preserves developmenttimel
- this timeline can be used to
  - discover which change caused a regression
  - understand why the code was changed when and where
  - understand who wrote which part of the code
  - keep a clear track of who has copyright on which part
- It is important to keep revision control system c lean
  - never commit two unrelated changes as one changeset
  - never commit without meaningful description/changelog

Classic Revision control systems

- RCS (Revison Control System)
  - per-file revision control
  - used in the 'old days', no network support
  - sometimes still used by sysadmins for local config files
- CVS (Concurrent Versioning System)
  - network-enabled version of RCS
  - supports checkin/commit of entire trees of files (not atom ic)
  - revisions are kept per-file
- SVN (Subversion)
  - revisions are for the entire tree!
  - much faster/better/modern, WebDAV based

Distributed Revision control systems

- git
  - specifically developed by Linux kernel develoeprs for kernel development
  - quite new, but very popular in the Linux world
  - based very simple primitives with toolkit on top
  - suports local and remote branches
  - keeps track of author and committer name/email
- mercurial/hg
- bazaar/bzr
- monotone/mtn
  - other systems, not discussed here

Working with diff

- the 'diff' program describes changes between two text files
- most commonly, the 'unified diff' (diff -u) is us ed
  - the output is human-readable, all developers can read it
- recursive operation for entire trees (diff -r)
- optionally ignore whitespace changes (diff -w)

Working with Changesets

- What is a Changeset?
  - A changeset is a specific logical change to software source code
  - A changeset is usually a patch (unified diff) plus decscript ion
  - A chronologic timeline of changesets is what your revision control system keeps
- Please always specify against which base version you made your changeset.
- Most of the time patch == changeset == diff

Contributing to FOSS projecst

- We never send entire versions of our program around
- We always use changesets (unified diff plus description)
- Distributed development works by sending around changesets by e-mail
- Mailinglists play important role so everyone can keep up-to-date with other people's changest
- The project/subsystem maintainer picks changesets from e-mail and applies them to his tree
  - Sometimes, maintainer can 'pull' changes from contributors' tree into hist tree
- The project/subsystem maintainer sends 'pull request' to higher maintainer

Lifecycle of a patch

- Lifecycle of a netfilter/iptables patch
  - Developer sends patch+description to netfilter-devel list
  - Other developers see it and may discuss it
  - After some review, a new version is sent to the list
  - The netfilter maintainer applies the patch to his tree (netf ilter.git)
  - At some point, the maintainer sends pull-request to network maintainer
  - Network-maintainer pulls the changes into his tree (net-2 .6.git)
  - At some point, the network maintainer sends pull-request to Linus
  - Linus pulls those changes during the next merge window into linux-2.6.git

General Rules

- make sure your code is compliant with Documentation/CodingStyle
- make sure your code is written against the lastest mainline git tree
  - sometimes, development against a specific subsystem git tree
- make sure your code passes the 'checkpatch.pl' script without errors
  - sometimes, warnings are acceptable. errors are never acceptable
- make sure you have read Documentation/SubmittingPatches

Don't do this

- Don't do this
  - reimplement code that already exist in the kernel (e.g. crc32)
  - include a protocol stack in your driver
    - protocol stacks (SD/MMC, 802.11, bluetooth) are vendor/device independent shared code
  - submit an OS independent driver with glue layer for Linux API's
  - submit drivers with support for older kernel API's (LINUX_VERSION_CODE)
  - submit drivers that include firmware in some header file
    - rather, use request_firmware() API to load firmware from filesystem
  - submit one driver for two completely different chips
  - submit two drivers for two chips that are 90% identical
  - submit drivers that don't work with latest linux-2.6.git

What's Signed-off-by ?

- The 'developer certificate of origin'
- If you add that line, you certify that you have
  - written the code yourself
  - and/or have permission to release it under GPLv2
- The idea is to keep track of who has written co de
- Maintainers usually add their signature, too
- See Documentation/SubmittingPatches

To which list should I send

- check the linux-2.6/MAINTAINERS file for 'L:' columns
  - or search on the project/subsytem homepagepage
  - if no specific list is found, use linux-kernel (lkml)
- for 'merge request' patches, Cc the maintainer
  - search for 'M:
- some list restrict posting to list subscribers, so you first need to subscribe
  - usually there is a web-based interface for subscription
  - sometimes you have to use e-mail based method

I sent the patch, what next?

- in the worst case, you get no feedback
  - if there's no feedback for one week, re-post and/or
  - send private mail to maintainer pointing out no feedback
- in the 'best' case your code gets merged immedi ately
  - you usually receive e-mail from the maintainer about it
- in the regular case, you get some feedback / change requests
  - try to answer to all questions as fast as possible
  - try to accomodate change requests as fast as possible
  - re-submit after integrating all change requests

My patch got merged, what next?

- if you wrote an entire driver and merged it
  - you 'own' the code, i.e. you should maintain it
  - you should send bug fixes and updates, one-by-one, as pat ches
    - don't wait for some "official release" !!!
  - it is your responsibility to make sure the code in mainline is synchronized
  - you will get Cc'ed by other people who want to change your driver
    - i.e. if some API change affects your driver
    - i.e. if somebody discovers a bug in your driver
    - you should verify the new code works and provide feedback
    - always keep the mailinglist in Cc

How to use git

- please see the practical demonstration

Thanks

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

Thanks for your Attention