Linux Kernel Development: Getting Started

Randy Dunlap

Linux Kernel Developer,
Maintainer, Mentor, and Janitor

FreedomHEC
May, 2006

Agenda

- Timetable: began life as a 3-hour tutorial
- Just hitting highlights today

- Abstract:
- Linux development is fast-paced and [as they say in Oregon] “things are different here.” This tutorial introduces some of the Linux culture and how to succeed when working with the Linux development community.

Topics

- Open source development style, values, culture
- Linux rapid development cycle
- Linux “maintainers” and hierarchy
- Communications methods
- Advantages of having a driver in the mainline kernel tree
- Coding style
- How to submit Linux kernel patches
- Some best known practices
- Legal/Licenses
- Testing
- Working in the Linux kernel tree

Major Goals

- Encourage new device driver development and support
- Driver code merged and maintained in mainline (GPL)

Development Style, Values, and Culture

- Learning curve, things are different
- Meritocracy – good ideas & good code are rewarded
- Chance to work on a real OS – any parts of it that interest you
- Massive amounts of open communication via email, IRC, etc.

Linux Culture

- Work in open, not behind closed doors (in smoke-filled rooms) #
- Community allegiance is very high
- Do what is right for Linux
- Meritocracy: good ideas and good code are rewarded
- Often driven by ideals and pragmatism, bottom-up development
- Not driven by marketing requirements
- Don't just take, give back too: #
  - Modifications are & remain GPL (if distributed)
  - Payment in kind, self-interest
  - Improve software quality, features used/understood more

Linux Culture (2)

- Committed to following and using standards (e.g., POSIX, IETF)
- Committed to compatibility with other system software
- Informal design/development: Not much external high-level project planning or design docs (maybe some internally at companies); can appear to be chaotic
- New ideas best presented as code, not specifications or requirements
- RERO: Release Early, Release Often -- for comments, help, testing, community acceptance #
- Possible downsides: flames, embarrassment

Linux Culture (3)

- Development community is highly technical
- Motivated and committed, but since many are volunteers, treat them with respect and ask/influence them, don't tell
- Continuous code review (including security)
- Continuous improvement
- Have fun!! :)

- **Follow the culture**

Linux Development Values

- Scratch your own itch
- Weekenders -> big business
- Code, not talk
- Pragmatism, not theory
- Thick skin
- Code producer makes [most] decisions
- Pride, principles, ethics, honesty
- Performance
- Hardware & software vendor neutral
- Technical merit, not politics, who, or money
- Maintainability & aesthetics: clean implementation, not ugly hacks (coding style)
- Peer review of patches (technical & style)
- Contributions earn respect

Some Things to Avoid

- Patents, binary modules, NDA
- Proprietary benchmarks
- Huge patch files
- Adding more IOCTLs
- Marketing
- Design documents
- Mention of accomplishments outside of the open source world
- No patch rationale
- How do I intercept a system call (or replace a syscall table entry)?
- Making demands instead of requests
- This {driver / feature} must be merged, it's important to our company.
- Date or release version requirements

Some Good Terms to Use

- Simpler
- Deletes N lines of code
- Faster (with data)
- Smaller (with data)
- Here's the code....
- Series of small patches....
- Tested... (how many configs)
- Builds on 8 architectures

When New Infrastructure Is Needed

- If a driver needs some new general-purpose subsystem infrastructure, don't try to merge it into the driver – that will be rejected
- Work with others (on m-l) to define and implement new infrastructure
  - Multipath I/O (MPIO)
  - SCSI transport services
  - Wireless LAN stack
  - RAID ??
  - FC State of the Union: http://lwn.net/Articles/132579/
- Driver developers can have an impact on kernel infrastructure

Drivers for New Hardware

- If your company wants to develop a GPL driver and merge it into Linux mainline, that's great news. Work with the development community (on public mailing lists) to accomplish that goal.
- Short of that, if your company can make hardware interface specs public and hardware available, there's a good probability that someone in the development community will develop a GPL driver for it.
- Short of that, make the hardware interface specs available privately to someone, but allow them to develop and publish a GPL driver.

New Driver Development

- Requires 1+ dedicated full-time software engineer to keep up with mailing lists and kernel changes, stay current, become a part of the development community
- This is a continuous, ongoing commitment, not an infrequent cameo appearance.
- Submit drivers for mainline inclusion and acceptance, not to distros. Major distros now require progress toward mainline acceptance.
- RERO for testing in the wild (“community”), in your lab, and at the distros

Development Cycle

- Moved from split “stable” (2.even) and “development” (2.odd) trees – caused delay and backport mania
- Now accepting development patches into the -mm patchset and moving them to the mainline kernel tree after a shakeout period (e.g., 2.6.11-mm3)
- 2.6.x kernel version cycle: make patches against Linus's tree (unless they only apply to some other tree or patchset)
- Time between 2.6.x releases, intermediate 2.6.x-rcN
- Nightly snapshots; automated builds of releases; commits mailing list
- 2.6.x.y stable kernel patches

Linux 2.6 Kernel Tree & Branches

mainline

2.6.11 2.6.12-rc1 2.6.12-rc2 2.6.12-rc3 ..... 2.6.12

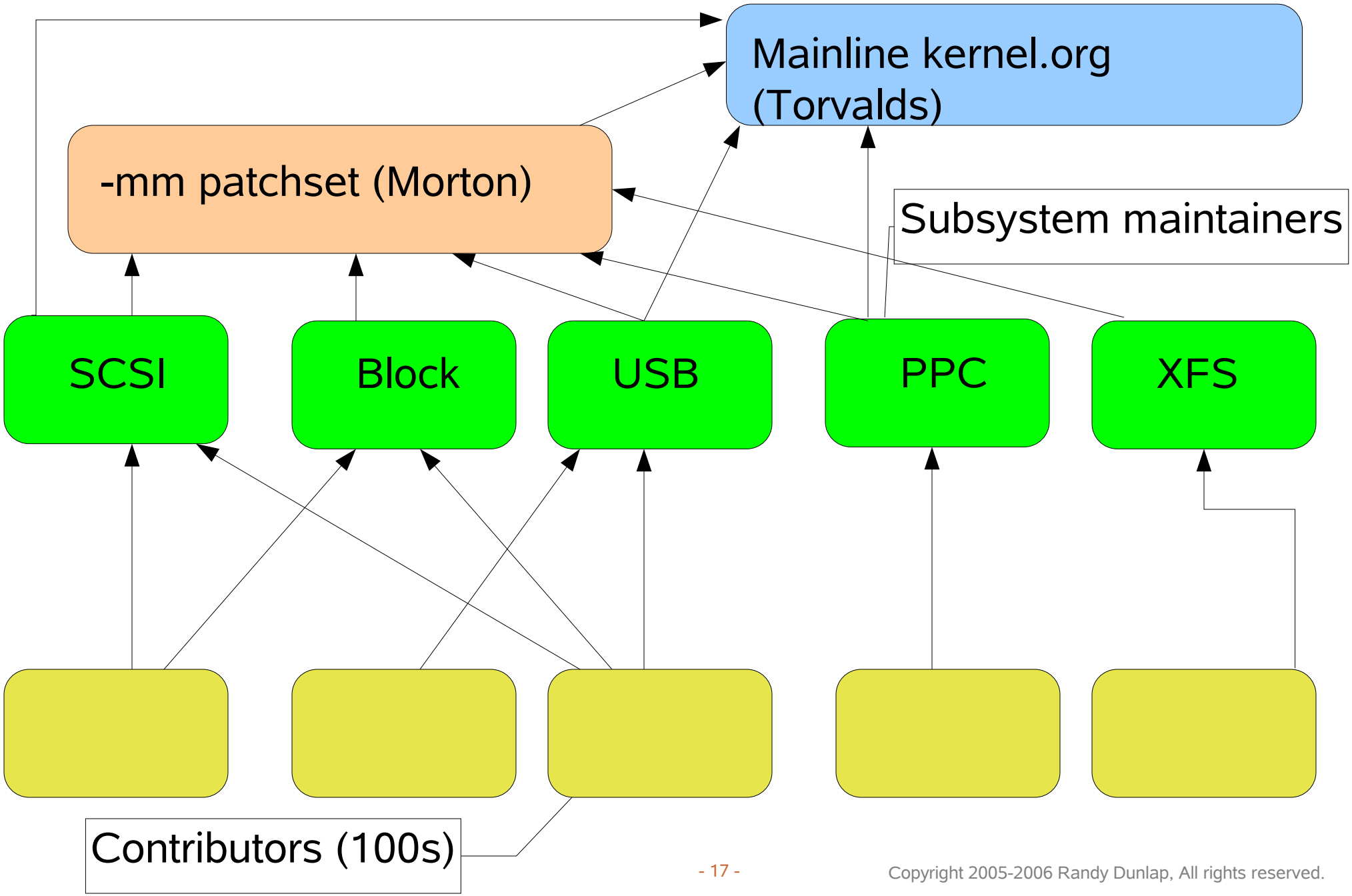
-mm patchset: review/test here before merge into mainline

-mm1 -mm1 -mm1 -mm1 -mm1
-mm2 -mm2 -mm2
2.6.12-rc3-mm3

stable patch series

2.6.11.1 2.6.11.2 2.6.11.3 ..... 2.6.11.9

Merges to Mainline (with exceptions)



Development Cycles

- Rapid development cycle, no timelines/schedules
- Only online documentation has a chance of being up-to-date
- Accommodate large changes and high rate of change without regressions
- Open discussion (mailing lists, archives, not private) #
- RERO, facilitates testing on a large variety of platforms #
- Maintainers available and accessible, don't disappear for long periods of time
- Test suites
- Bug tracking

Rates of Kernel Change

- first six months of 2.4 devel: -220,000 lines, +600,000 lines
- first six months of 2.6 devel: -600,000 lines, +900,000 lines
  - 1.5M lines changed in a 6.2M line tree
  - 64 MB diff in six months - and that's the stable kernel
- Current 2.6.11 -> 2.6.12-rc4 (10 weeks): 729 K lines, 22 MB diff
- Current 2.6.12-rc4-mm1 patchset: 414 K lines, 13 MB diff

Topics

- Open source development style, values, culture
- Linux rapid development cycle
- Linux “maintainers” and hierarchy
- Communications methods
- Advantages of having a driver in the mainline kernel tree
- Coding style
- How to submit Linux kernel patches
- Some best known practices
- Legal/Licenses
- Testing
- Working in the Linux kernel tree

Maintainers and Hierarchy

- Loose hierarchy with “benevolent dictator”
- Kernel series maintainers (2.6) – Linus and Andrew Morton
- Patch (“stable”) maintainers (2.6.x.y) – Greg Kroah-Hartman and Chris Wright
- Top-level maintainers are gatekeepers, integrators, tiebreakers or overrulers when needed
- Delegate to lieutenants and individual maintainers; share the load
- Strong trust system -> begin with small patches for credibility
- Maintainers don't have absolute authority

Maintainers & Hierarchy (2)

- Kernel Janitors, security kernels, some embedded support
- Arch and subsystem maintainers: coordinate subsystems and maintain consistency
- Driver maintainers: cover all current mainline kernels and update to new kernel APIs, even development APIs
- See files: linux/MAINTAINERS and linux/CREDITS

Communications

- Communicating is hard, let's go shopping
- Writing ideas/thoughts down is good (but too wordy may be ignored)
- Participate constructively
- Mailing lists & archives (newsgroups)
- Working in open/public (technical readers/writers) vs. embarrassment
- Discussion and decisions on lists, no meetings required
- Work through concensus (with exceptions)
- Project web pages, IRC channels
- Developer conferences

Mailing List Etiquette

- Use Reply-to-All, threaded (Message-ID, References)
  - > > Try A or B.
  - > I prefer A, sound OK?
  - yes
- Be prompt with replies (being responsive is important)
- No encoded or zipped attachments (inline preferred, text/plain attachments OK); others are often ignored
- No HTML or commercial email, no auto-replies (OOO/vacation)
- ALL CAPS == SHOUTING
- Use < 80-column width lines (70-72 is good) for text (not for patches)

Mailing List Etiquette (2)

- Keep it technical and professional. If attacked (flamed), stick with technical points, don't get involved with attacks, & move on.
- Trim replies (body) to relevant bits (don't modify To:/Cc: recipient list).
- Don't cross-post to closed mailing lists.
- Non-English speakers
- http://www.arm.linux.org.uk/armlinux/mletiquette.php
- RFC 1855: Netiquette Guidelines: http://www.ietf.org/rfc/rfc1855.txt

Mailing List Etiquette: No top-posting

- A: http://en.wikipedia.org/wiki/Top_post
  Q: Where do I find info about this thing called top-posting?
  A: Because it messes up the order in which people normally read text.
  Q: Why is top-posting such a bad thing?
  A: Top-posting.
  Q: What is the most annoying thing in e-mail?

  A: No.
  Q: Should I include quotations after my reply?

Mailing Lists

- Most lists have spam filters [to get past]; you probably need to use them also
- LKML a.k.a linux-kernel (@vger.kernel.org)
- LKML FAQ at http://www.tux.org/lkml/
- Index: http://vger.kernel.org/vger-lists.html and their archives
- Kernel patch commits mailing list: git-commits-head@vger.kernel.org

More Kernel Project Mailing Lists

- Networking development: netdev@vger.kernel.org
- Index: http://oss.sgi.com/ecartis/
- Subsystems: arches, filesystems, MM/VM (http://www.linux-mm.org), security, drivers (ACPI [SF.net], I2C, IDE, video, PCI, PCMCIA, IEEE 1394 [SF.net], USB [SF.net], SCSI, Infiniband, Bluetooth)
- More mailing lists in MAINTAINERS file and at http://kernelnewbies.org

Mailing Lists for Linux Starters

- http://kernelnewbies.org - kernelnewbies@nl.linux.org
- http://janitor.kernelnewbies.org - kernel-janitors@lists.osdl.org
- os_drivers@lists.osdl.org
- kernel-mentors@selenic.com
- Trivial patch monkey: trivial@kernel.org and http://www.kernel.org/pub/linux/kernel/people/bunk/trivial

- http://vger.kernel.org/majordomo-info.html - has list info and taboos
- Kernel announcements: linux-kernel-announce@vger.kernel.org

Mailing List Archives

- Archives for almost all
  - http://gmane.org has interface
  - http://marc.theaimsgroup.com/ has many, with Search
  - http://lkml.org -- kernel list only
  - Google groups
- http://www.kerneltraffic.org/ -- summaries
- http://lwn.net/ -- summaries

Project Web Pages

- SourceForge.net (http://sf.net): web pages, mailing lists, CVS, bug tracking, etc.
- OSDL: http://lists.osdl.org - http://developer.osdl.org - http://bugme.osdl.org == http://bugzilla.kernel.org
- Hardware vendors: IBM, HP, Dell
- Distro vendors (Red Hat, SUSE, Debian)

Development Conferences

- Linux Symposium (Ottawa, July)
- Linux Conference AU (LCA, usually March-April)
- LinuxTag (Germany, June)
- Linux Kongress (Germany, September)
- Kernel (July), GCC (June), Desktop (July) summits
- Focused mini-summits (networking, power management, storage management, filesystems, wireless, desktop)

Related Documentation

- lwn.net articles: http://lwn.net/Articles/driver-porting/
- LDD3 book: http://lwn.net/Kernel/LDD3/
- Driver "DOs and DON'Ts": at the KJ web site
- Arjan: How Not to Write a Driver (OLS, at KJ web site)
- Greg (PCI, USB maintainer): Coding Style, Writing Portable Code, et al (http://www.kroah.com/linux/)
- Andrew (top kernel maintainer): TPP: The Perfect Patch: http://www.zip.com.au/~akpm/linux/patches/stuff/tpp.txt #
- Jeff (net drivers maintainer): http://linux.yyz.us/patch-format.html #

Why Merge Into the Mainline Kernel Tree

- Background on kernel API/ABI
  - Kernel API is not stable; no kernel binary API (ABI)
  - A static (stable) API limits innovation and adds “cruft”
- Userspace API is very stable and will remain so
- Interfaces and structures depend on toolchain & kernel config options and distro changes, so single kernel ABI isn't feasible
- Old interfaces are removed (sometimes after a “deprecated” grace period), preventing their continued use which could cause system outages and kernel bloat
- See file: linux/Documentation/feature-removal-schedule.txt

Advantages of Merging into Mainline (1/3)

- Keeps the driver updated and working, even if its maintainer disappears or the OEM stops supporting/updating it
- Kernel API changes are merged for you: performance improvements, bug fixes, security fixes, parameter or structure changes
- Kernel changes increase quality of driver while maintenance costs to the maintainer decrease (are amortized)
- Other people will add features to your driver
- Others will find & fix bugs in your driver
- Others will find & fix performance/tuning opportunities

More Merge Advantages (2/3)

- **Driver is automatically shipped in all Linux distros without having to ask distros to merge & ship it so all stay in sync**
- Driver is available for use on 20+ CPU architecutures, not just a handful [still requires proper endian handling; check with 'sparse']
- Driver get broader testing and review
- **Driver maintainer is relieved from maintaining external patchsets – difficult even if open-source code**
- Offers a uniform feature set to all users
- Becomes the de facto driver (with you as Maintainer), keeping work focused on one driver

Merge Advantages (3/3)

- Several large distro vendors require “upstream” progress (e.g., public reviews on mailing lists)
  - Merging via distros can lead to incompatibilities with mainline
- Discourages mini-forking & fragmentation: bad for users (different features & bugs) & for the fork maintainer
  - Users with non-mainline drivers can end up helpless or unsupported or locked into one distro

Disadvantages of Merging

- Must adapt code to kernel coding style
- Must go thru peer review and respond to feedback, make changes
- Remove compatibility layers, old kernel version support, other OS support
- May need to make it arch-portable (endianness, word sizes)
- May need design changes or features added
- Probably will take several weeks of posting patches, feedback, more changes, but that's a one-time thing
- **Cost of not listening: invest man-years in development then told "the architecture is wrong, redo it"**

Merge to Mainline (summary)

- Big effort to use mainline public kernel for merging
- Keeps all distro vendors the same
- **Provides for more and better testing, review, and bug-tracking**

Topics

- Open source development style, values, culture
- Linux rapid development cycle
- Linux “maintainers” and hierarchy
- Communications methods
- Advantages of having a driver in the mainline kernel tree
- Coding style
- How to submit Linux kernel patches
- Some best known practices
- Legal/Licenses
- Testing
- Working in the Linux kernel tree

Coding Style

- **Clean code, not for other OS-es or for other Linux versions**
- Use comments, but not for obvious code; on data structures
- Drivers, filesystems, etc., are not arch-specific (must be arch-portable)
- Follow style in surrounding code
- Very minimal use of typedefs (only for basic types)
- Minimize use of #ifdef in C source files, use stubs in header files instead (as much as possible/feasible)
- Don't use #ifdefs to support multiple kernel versions
- Documentation/: CodingStyle, SubmittingPatches/Drivers, & web pages

Coding Style (2)

- Don't abuse the kernel API
- Simpler is better (“eschew obfuscation”)
- Minimize macro usage (prefer inline functions for type-checking)
- Stubs: include/linux/highmem.h, init.h, module.h, sched.h, swap.h, include/asm-generic/dma-mapping.h
- Linux kernel is written in C, not C++
- Use /* ... */ for comments (not //)
- Function comments in “kernel-doc” style
- Use (but don't abuse) 'goto', especially for error handling (one function exit path) [and undo allocations etc. in error handling]

Coding Style (3)

- Use C99-style struct initializers
- Use tabs for indentation, not spaces (Tab size is 8)
- Don't disable or ignore compiler/build warnings
- Use 'sparse' for even more warnings
  - $ make C=1 ...
- Use 'make checkstack', 'make buildcheck', 'make namespacecheck' to check for details
- Don't make functions or data global unless needed (mostly 'static')
- Don't use deprecated kernel APIs
- Don't use anonymous unions (gcc 2.9x tool problem)

Coding Style (4)

- Avoid 'extern' in C files, use headers instead
- #include file order
  - <linux/file.h> (alphabetically when possible)
  - <asm/file.h> (alphabetically when possible)
  - “localfile.h” (alphabetically when possible)
- Don't #include files unless they are needed/used

Coding Style (5) (Policies)

- Don't init static or global data to 0 (it's all cleared during init)
- Initialize data statically instead of during init run-time if possible
- Don't abuse the kernel stack (it's small)
- Don't use recursion (sometimes OK if it has a low bound)
- Push data conversions (like graphics) to userspace
- For locking (mutexes, critical regions), don't use 'volatile'; analyze and use locks or semaphores
- Don't use or depend on BIOS calls or data except during kernel init, and then as little as possible

Coding Style (6) (Policies)

- Don't add IOCTLs, use /sys (sysfs)
- Don't trust data coming from userspace
- Don't read/write files from kernel space (exception: firmware downloads)
- Check that code compiles UP/SMP and MODULE/not MODULE and on multiple arches if applicable and possible (OSDL PLM will do 8 arch. cross-compiles of one patch)

How to Submit Linux Kernel Patches

- Patch -current mainline from kernel.org or -mm patchset
- Send patches to subsystem maintainer, driver maintainer, & mailing list #
- Each patch (re-)submission should include feature justification and explanation, not just the patch #
- Use the DCO ("Signed-off-by: Your Name your.name@example.com") #
- Patches should be encapsulated (self-contained) as much as possible, not touching other code (when that makes sense) #

Submitting Patches (2)

- ONE patch per email, logical progression of patches, not mega-patches, not attached and not zipped (cannot review/reply) #
- Don't do multiple things in one patch (like fix a bug and do some cleanup)
- Check your email client: send a patch to yourself and see that it still applies (doesn't damage whitespace, line breaks, content changed) before going public with it
- Patch must apply with 'patch -p1'; i.e., use expected directory levels
- Don't use PGP or GPG with patches, they mess up patch scripts

From: Randy Dunlap <rdunlap@xenotime.net>

register_chrdev() can return errors (negative) other then -EBUSY,
so check for any negative error code.

Signed-off-by: Randy Dunlap <rdunlap@xenotime.net>
---
 drivers/pcmcia/ds.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

```
diff -Naurp ./drivers/pcmcia/ds.c~ds_check_major ./drivers/pcmcia/ds.c
--- ./drivers/pcmcia/ds.c~ds_check_major      2005-05-12 13:16:41.000000000 -0700
+++ ./drivers/pcmcia/ds.c             2005-05-12 19:45:36.000000000 -0700
@@ -1592,9 +1592,9 @@ static int __init init_pcmcia_bus(void)

                /* Set up character device for user mode clients */
                i = register_chrdev(0, "pcmcia", &ds_fops);
-               if (i == -EBUSY)
+               if (i < 0)
                        printk(KERN_NOTICE "unable to find a free device # for "
-                           "Driver Services\n");
+                           "Driver Services (error=%d)\n", i);
                else
                        major_dev = i;
```

Some Best-Known Practices

- Track origin(s) of your software (COO: Certificate of Origin)
- User DCO (Developer's Certificate of Origin) for kernel contributions
- Management approval and legal clearance to submit source code
- Some companies may require a Waiver of Copyright
- Send patches directly to their intended maintainer for merging (they don't troll mailing lists looking for patches to merge)
- Copy patches to the appropriate mailing list(s), not private (don't work in isolation)
- Subscribe to relevant mailing lists (or use one representative for this)
- Listen to review feedback and promptly respond to it

Best Known Practices (2)

- Linus normally does not acknowledge when he merges a patch
- Use correct 'diff' directory level (linux/ top-level directory) and options (-up)
- Use source code to convey ideas
- Generate patch files against the latest development tree branch (-rcN) or mainline kernel if there is no current development branch
- Make focused patches or a series of patches, not large patches that cover many areas or that just synchronize a (CVS) repository with the kernel source tree
- Use the available docs.

Best Known Practices (3)

- Include Copyright and license: MODULE_LICENSE("GPL");
- Use an email client that supports inserting patches inline (not as attachments)
- Begin with small patches: use kernel-janitor m-list
- For larger patches or complete drivers or features, use the kernel-mentors m-list (for beginner feedback/comments/corrections)
- Don't misuse (abuse) the kernel API; e.g., avoid "void *" function arguments
- Don't post private email replies to a public m-list (without permission)
- Don't introduce gratuitous whitespace changes in patches

Best Known Practices (4)

- Back up your patch with performance data (if applicable)
- Don't add binary IOCTLs unless there are no other acceptable options; use sysfs (/sys) or private-fs or debug-fs or relayfs or netlink if possible
- Make Linux drivers that are native Linux drivers, not a shim from another OS
- Don't introduce kernel drivers if the same functionality can be done reasonably in userspace
- Try to be processor- and distro-agnostic (except for CPU-specific code)
- Don't be afraid to accept patches from others

Best Known Practices (5)

- Keep your patch(es) updated for the current kernel version
- Resubmit patches if they are not receiving comments
- Release early, release often
- Open, public discussion on mailing lists
- One patch per email
- Large patches should be split into logical pieces and mailed as a patch series
- Make testing tools available & easy to use; your device(s) will get better testing
- Giving hardware to developers can result in drivers written for you

Topics

- Open source development style, values, culture
- Linux rapid development cycle
- Linux “maintainers” and hierarchy
- Communications methods
- Advantages of having a driver in the mainline kernel tree
- Coding style
- How to submit Linux kernel patches
- Some best known practices
- Legal/Licenses
- Testing
- Working in the Linux kernel tree

Legal & License Points

- IANAL, get legal advice
- Open source is a business decision, free software is an ethical one
- http://www.opensource.org - Open Source Initiative, many open source licenses listed, but desire is to significantly reduce to number of licenses that are used
- Track origins of your software used internally in development
- Use DCO for Linux kernel contributions #

Legal & Licenses -2

- All distributed kernel modules must be open-source, GPL-compatible licensed (dual)
- EXPORT_SYMBOL(), EXPORT_SYMBOL_GPL(), and EXPORT_SYMBOL_GPL_FUTURE()
- GPL exports discussion: http://www.kerneltraffic.org/kernel-traffic/kt20021021_18

Bug Reporting and Tracking

- kernel bugs database: fix bugs or help update info:
  http://bugzilla.kernel.org or (same)
  http://bugme.osdl.org
- Mailing lists are heavily used for bug reporting
- Sourceforge.net project pages : some projects use this bug tracker
- Other project-specific bug tracking

Kernel Test Projects

- LTP: http://ltp.sourceforge.net
- Open POSIX test suite: http://posixtest.sourceforge.net
- OSDL PLM for building and cross-building patches: http://www.osdl.org/plm-cgi/plm/
- OSDL STP framework and servers: http://www.osdl.org/stp/
- http://test.kernel.org frequent tests & reports

Virtualization for Kernel Testing

- UML for testing
- Virtualization for testing --Linux virtualization summary: http://www.linuxsymposium.org/proceedings/reprints/Reprint-Wright-OLS2004.pdf
- XEN, qemu, Bochs (x86)

Working in the Linux Kernel Tree

- About 18,000 source files
- core functionality: kernel/, mm/, init/, ipc/, lib/
- drivers/, fs/, net/
- arch/, security/, crypto/, ...

Kernel Config

- Generate or edit config with any of:
  make {menuconfig, xconfig, gconfig, config, defconfig, oldconfig}
- Build kernel: make all
- Install kernel:
  - 1: su to root
  - 2: make install
  - 3: make modules_install
  - 4: edit LILO config (+ run lilo) or edit GRUB config
  - 5: reboot

Patch Maintenance Tools

- Use 'diff' to create patches (even for complete new files or to add or remove files)
- Manual diff-ing:
  - Can diff complete unmodified tree vs. a modified tree
  - Can diff one or a few modified files vs. their original files
  - Use 'gendiff' or 'genpatch' to generate patchsets
- Use 'patch' to apply patches that you create or receive
- 'patch-kernel' to update kernel directory in place

Patch Tools (2)

- Can use 'patch-scripts' or 'quilt' for patch management
  - http://www.zip.com.au/~akpm/linux/patches/patch-scr
  - http://savannah.nongnu.org/projects/quilt
- Send a series of patches (e.g.):
  http://www.speakeasy.org/~pj99/sgi/sendpatchset
  or similar script in patch-scripts
- SCMs: your choice, flavor of the day
- 'git' for kernel source code management:
  http://www.kernel.org/git/

References (1/3)

- http://lwn.net/Articles/driver-porting/
- http://lwn.net/Kernel/LDD3/ - Linux Device Drivers 3rd ed.

  ***** subscribe to LWN.net *****
- http://kernelnewbies.org - articles, documents, scripts, book recommendations, beginner Q&A, IRC, mailing list
- http://janitor.kernelnewbies.org - docs, scripts, Dos/DONTs, TODO list, IRC, mailing list
- http://www.linuxsymposium.org/2005/ - OLS proceedings

References (2/3)

- http://www.kroah.com/linux/ - conference slides, papers, talks, tools, coding style, development process, dealing with kernel community, writing portable kernel code
- http://people.redhat.com/arjanv/olspaper.pdf - How to NOT write kernel code – actual examples (OLS 2002)
- OLS 2004 keynote, Andrew Morton: http://www.zip.com.au/~akpm/linux/patches/stuff/ols-200

References (3/3)

- Linux kernel source tree:
- linux/ MAINTAINERS, CREDITS
- linux/Documentation/
  - CodingStyle
  - SubmittingPatches
  - SubmittingDrivers
  - feature-removal-schedule.txt (deprecated)
  - stable_api_nonsense.txt
  - SubmitChecklist (currently only in -mm patchset)
  - HOWTO (do kernel development)

Credits

- Hugh Blemings
- James Bottomley
- Matt Domsch
- Jeff Garzik
- Clyde Griffin
- Christoph Hellwig
- Gerritt Huizenga
- Greg Kroah-Hartman
- Pat Mochel
- Andrew Morton
- Arjan van de Ven
- Ric Wheeler
- Cliff White
- Chris Wright
- Top-posting A&Q from a .sig on the old crackmonkey m-l