What happens when kernel and userland don't talk?

linux.conf.au 2009
Hobart, Australia; 22 Jan 2009

Michael Kerrisk
Fellow, Linux Foundation
mtk.manpages@gmail.com

http://www.kernel.org/pub/linux/docs/manpages
http://linux-man-pages.blogspot.com

Overview

- What's the problem?
- Examples
- Causes
- How can we improve things?
- Are things getting better?
- How to help
- Conclusion/Questions/Discussion

What's the problem?

Background: the kernel-userspace interface

- Kernel presents a programming interface to userspace programs:
  - system calls (generally mediated by glibc)
  - `/proc` files
  - *ioctl()* (special-case, catch-all system call)
  - and various others: netlink, sysfs, configfs, etc.

Background: Application Binary Interfaces

- Kernel defines an Application Binary Interface (ABI)
- ABI == set of rules describing how two binaries exchange data
- Defines, e.g.:
  - Registers and stack locations used for exchange
  - Meaning of exchanged binary values
- Programs above the ABI depend on ABI remaining stable
- If *existing* kernel ABI changes, (perhaps very many) userspace binaries will break
- Repeat after Linus: “We do not break the ABI”
- *Extensions* to kernel ABI usually viable

Proving the point

(Examples of kernel-userspace interface problems)

No time for testing; let's just ship it!

- *inotify(7)* `IN_ONESHOT` flag [2.6.13]
  - never worked
  - fixed in 2.6.16
- *utimensat(2)* [2.6.22]
  - Multiple bugs
  - Fixed in 2.6.26
- *signalfd()* [2.6.22]
  - didn't correctly obtain data sent with *sigqueue(2)*
  - Fixed in 2.6.25
- → pre-release testing is limited (or non-existent)
- (Bug fixes are ABI changes)

Breaking portability

- SUSv2 (1998) says successful *sched_setscheduler()* call should return previous policy
- Most systems do this
- But not Linux
- On Linux, 0 is returned on success
- Impossible to change now (don't beak ABI)

Sheer inconsistency

- Before Linux 2.6.9, no privilege was needed for *munlock()*, but *munlockall()* needed `CAP_IPC_LOCK`
- Some memory-related system calls round arguments to page boundaries
  - *mlock(addr, length)* rounds *addr* down, and *length* up
  - *remap_file_pages(addr, length, …)* rounds *addr* down, and *length* down too!
  - Can't fix (don't break ABI)
- When interfaces are inconsistent, programmers do the wrong thing

Let's code it now, then think about it (1)

- Vanishing arguments:
  - *readdir(2)* ignores *count* argument
  - *getcpu(2)* [2.6.19] ignores *tcache* argument
  - *epoll_create()* [2.6] ignores *size* argument since 2.6.8
- Either:
  - argument wasn't needed to start with; or
  - realized as a bad idea and made a no-op
- Probably, it should never have been there to start with
- Can't remove argument (don't break the ABI)

Let's code it now, then think about it (2)

- *tkill(tid, sig)* [2.5.4] signal thread in same thread group
  - Implement POSIX threads semantics (*pthread_kill()*)
  - But maybe *tid* was recycled; now in different thread group
  - Solution: *tgkill(tgid, tid, sig)* [2.5.75]
- *futimesat()* [2.6.16] intended to extend *utimes()*
  - Proposed for POSIX.1-2008
  - Implemented on Linux while standard was still in draft
  - POSIX.1 members realized API was still insufficient
  - So, POSIX.1-2008 standardized different API
  - *utimensat()* added in Linux 2.6.22
- Can't remove these interfaces (don't break the ABI)

Is anyone looking at the big picture?

- Capabilities
- Divide *root* into *small*, distinct pieces

  CAP_AUDIT_CONTROL CAP_AUDIT_WRITE CAP_CHOWN CAP_DAC_OVERRIDE
  CAP_DAC_READ_SEARCH CAP_FOWNER CAP_FSETID CAP_IPC_LOCK CAP_IPC_OWNER
  CAP_KILL CAP_LEASE CAP_LINUX_IMMUTABLE CAP_MAC_ADMIN CAP_MAC_OVERRIDE
  CAP_MKNOD CAP_NET_ADMIN CAP_NET_BIND_SERVICE CAP_NET_BROADCAST CAP_NET_RAW
  CAP_SETFCAP CAP_SETGID CAP_SETPCAP CAP_SETUID **CAP_SYS_ADMIN** CAP_SYS_BOOT
  CAP_SYS_CHROOT CAP_SYS_MODULE CAP_SYS_NICE CAP_SYS_PACCT CAP_SYS_PTRACE
  CAP_SYS_RAWIO CAP_SYS_RESOURCE CAP_SYS_TIME CAP_SYS_TTY_CONFIG

- Great idea: minimizes risk if program is compromised
- But which capability do I (an implementer) use?
- Ahh! I know!
- `CAP_SYS_ADMIN`, the new *root*, 192 uses in 2.6.29-rc

Did we really mean to do that?

- Linux 2.6.9 added *waitid()* (extension of *waitpid()*)
- POSIX.1 specifies:
  - Application shall ensure that *infop* argument is not `NULL`
  - System shall return 0 if *waitid()* is successful
- On Linux, if *infop* is `NULL`, *waitid()* succeeds, and returns the PID of waited-for child
- A non-standard, non-portable, unintended, unrequested, and undocumented feature
- Linus calls it an extension
- I call it an accident
- Should we fix it?

Did these problems need to happen?

- Most of these problems should never have happened
- It's not the developers' fault – okay, maybe it is sometimes
- The real problem is processes and structures
- If only we had had:
  - More, better, earlier testing
  - More, better, earlier design review

Causes

Some possible causes

- The API-development process
- Distributed ownership of the API
- Communication

API development process: The Ideal(?)

1. Canvas users about needs for API
2. Write a specification of API (e.g., man page)
3. Code up initial version of API
4. Widely publicize proposed API, publish kernel patch, provide example programs and documentation
5. Get review comments about design, revise if needed
6. Thoroughly test API; fix bugs
7. Write API documentation, get editing and review
8. Interface is accepted into main-line kernel

API development process: Common reality

1. A kernel developer has or hears of a good idea
2. Developer codes patch
3. Developer posts patch to LKML, with *brief* description
   - "I tested it"; commonly, little documentation or test code
4. *Often*, someone looks at code, suggests some fixes
5. *Usually,* no one else does any testing
6. Code sits for a while in *-mm* or another tree
7. Code gets accepted into mainline
8. Later: someone writes a man page
9. Later: find and fix all bugs
10. Later: realize API could have been better designed

Distributed ownership (1)

- One contributing factor for our problems is *distributed ownership* of kernel-userspace API
- Who owns the interface?
  - Ownership == determines what is/isn't added to interface
  - Kernel developers: implement the interface
  - Glibc: provides wrappers for many system calls
  - Users: can employ unintended features in API
  - Testers (e.g., LTP): write test specs and tests for API
  - *man-pages*: produces reference describing intended behavior of interface
- *Who owns the interface*, mtk, LPC, Sep 2008, Portland

Distributed ownership (2)

- Kernel developers, users, glibc, testers, documenters all have some legitimate claim to ownership of API
- But, coordination and communication between these groups is often poor, or sometimes even non-existent
  - This is the origin of many of our problems

Communication

- Significant problem!
- The different stakeholders often inhabit separate worlds, e.g.:
  - Many kernel developers seldom write userspace programs
  - Most userspace programmers don't read/write kernel code
  - Testing (LTP) often starts after implementation finalized
  - Documentation also sometimes starts only after implementation is finalized
- Relatively little interaction between these groups

How can we improve things?

Better communication and coordination

- Need more communication between various groups (kernel developers, userland programmers, testers, documenters, etc.)
- Needs to happen in development “real time”
- Better coordination between groups, so that, e.g.:
  - Developing LTP test suite starts during API development
  - Documentation is complete before API is finalized
  - Good design review occurs before final release

We need more people doing testing and review

Barrier to getting involved is high

- Reading kernel code is often hard
- Not always easy to find out about proposed changes
- Makes it hard for userland programmers and testers to get involved early in API design review and testing

Lowering the barrier

- Need to make understanding new APIs easier
- Usually: difficult to get relevant kernel developers and userspace programmers together
- Mailing lists are part of the solution
- But better, more, earlier documentation is also important

Documentation

- A structured, thought-through description of a new API
- Leads to self-review by developer
  - But, shouldn't be written (only) by developer
- Lowers barrier to understanding new API
  - ==> easier for others to start testing and design review
  - Bugs found earlier
  - Design faults found earlier
- Provides design guidance for future API designers
  - Consistency, integration, “good design models”
- Needs to be done as part of development process, not after API is finalized

Are things getting better?

Are things getting better? (1)

- Maybe, slowly
- Yes:
  - For the last many months, my work has been funded by Linux Foundation (but... more later)
  - Man pages now often written in parallel with new APIs
  - A *linux-api* mailing list now exists; should be CCed on all API changes, so stakeholders know about them
  - Some support for culture of “API patches should be accompanied by documentation”
  - Personally, I've managed to find more bugs and contribute to design review earlier in the API development process

Are things getting better? (2)

- And no
  - Large parts of API remain undocumented
  - Many bugs still don't get caught until after release
  - Much more work could be done on design review
  - Still lack enough people doing early testing, design review, and documentation

How to help

How to help – design review and testing (1)

How to help – design review and testing (1)

- Get involved in API design review and testing
- Find the mistakes of famous kernel developers!
- Quickly make a difference to quality of Linux
- Good lead in to getting involved in kernel development
- Sign up on *linux-api* mailing list
- Read the summary of user-visible API changes that usually appears on lwn.net at *-rc1* time
- Sign up on LTP mailing list

How to help – design review and testing (2)

- All you need to be able to do is
  - Find the patch
  - Apply kernel patches
  - Build a kernel
  - Write C test programs (often quite small)
  - Look at API and think about what could go wrong

How to help - documentation

- Get involved in documentation
  - *groff* is clunky, but basics are very simple
  - *man-pages(7)*
- Sign up on *linux-man*
- Review new man pages, for any of:
  - Spelling / typos
  - Clarity of explanation
  - Technical accuracy (write tests!)

How to help -

- Fund my job!

Concluding thoughts

- Kernel-userspace interfaces are contracts
- Cast in stone
- We live with them “forever”
- Need to get them correct, right from the start
- Getting things right:
  - Requires some degree of planning and coordination
  - Probably more than we currently do
- Linux may be evolution, but intelligent design might sometimes get us there better and faster

Thanks!

Note also Stephen Hemminger's talk:
“Kernel/Userspace interfaces: can we ever get it right?”,
today, 17:00, Social Science 1

Discussion / Questions

http://userweb.kernel.org/~mtk/papers/lca2009/kernel_and_userland_dont_talk.pdf

Michael Kerrisk - mtk.manpages@gmail.com

linux-api@vger.kernel.org

linux-man@vger.kernel.org

http://www.kernel.org/pub/linux/docs/manpages

http://linux-man-pages.blogspot.com