

# Why kernel space sucks

*(Or: An abridged history of kernel—user-space interface blunders...)*

**linux.conf.au 2013**

Canberra, Australia  
2013-02-01

Michael Kerrisk  
© 2013

<http://man7.org/>  
[mtk@man7.org](mailto:mtk@man7.org)  
[mtk@lwn.net](mailto:mtk@lwn.net)



# Who am I?

- Professionally: programmer (primarily); also educator and writer
- Working with UNIX + Linux since 1987
- Linux *man-pages* maintainer since 2004
  - 128 releases (as at Jan 2013)
  - written or cowritten ~300 of ~950 man pages
  - lots of API review and testing, many bug reports
- Author of a book on kernel—user-space API
- IOW: **Lots** of time looking at the interface

# Intro: Why user space sucks

- Paper/talk by Dave Jones of Red Hat
  - First presented at Ottawa LS 2006
- A lead-in to deconstructing a couple of myths
- *Why Userspace Sucks* → WUSS
  - <http://www.kernel.org/doc/ols/2006/ols2006v1-pages-441-450.pdf>
  - <http://www.codemonkey.org.uk/projects/talks/ols2k6.tar.gz>
  - <http://lwn.net/Articles/192214/>



# Motivation for WUSS

- We (kernel developers) have created a kernel that performs magnificently
- But, can we make it better?
  - Why does it take so long to boot, start applications, and shut down?
  - Why does my idle laptop consume so much battery power?

# Findings from WUSS

- Dave Jones instruments the kernel, and finds...
  - Boot up: 80k `stat()`, 27k `open()`, 1.4k `exec()`
  - Shutdown: 23k `stat()`, 9k `open()`
- User-space programmers wreck performance doing ***crazy*** things!
  - `open()` and reparse same file multiple times!
  - read config files for **many** devices not even present!
  - `stat()` (or even `open()`) 100s of files they never need
  - timers triggering regular unnecessary wake-ups



# Conclusions from WUSS

- Room for a lot of improvement in userspace!
- User-space programmers should be aware of and using trace and analysis tools
  - (perf, LTTng, ftrace, systemtap, strace, valgrind, PowerTOP, etc.)





Kernel space



User space

“We (kernel developers) are much smarter than those crazy user-space programmers”



Kernel space



User space

“User space developers are weenies. One of the most fun parts of git development for me has been how easy everything is ;)”  
[Git mailing list, 22 Jan 2008]







Kernel space



User space

Something's wrong with this picture...

Let's question a couple of myths...

- **Myth 1:** Kernel programmers (can) always get things right (in the end)
- **Myth 2:** Code is always the best way to contribute to Free Software



# Terms: API versus ABI

- **API** == Application **P**rogramming Interface
- **ABI** == Application **B**inary Interface
  - Conventions used for communicating between two binaries; for example:
    - size of arguments (in bytes),
    - meanings of numeric value of args
- **API** → **[compilation]** → **ABI**
- **ABI compatibility** == compatibility of **compiled** binaries
  - Changing ABI affects *existing* compiled binaries



# Myth 1

Kernel programmers  
(can) always get things right  
(in the end)

*Except, there's (at least) one place  
where they don't: the interface*



# The kernel—user-space interface

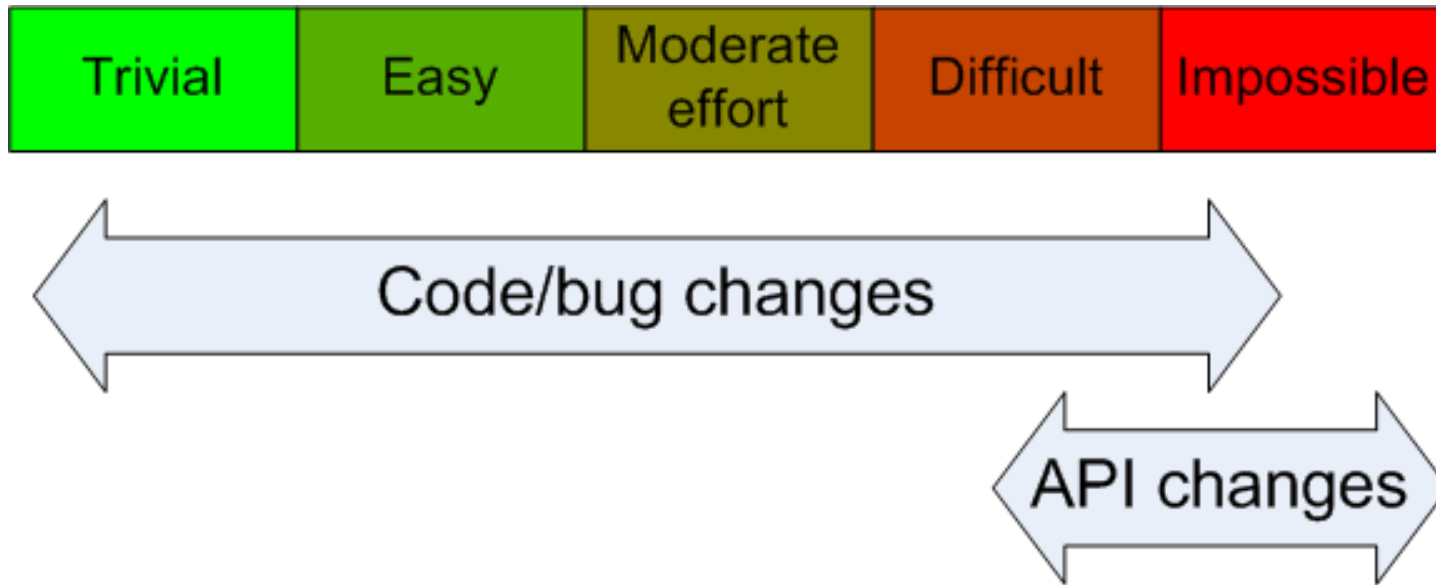
- Application programming interface (API) presented by kernel to user-space programs
  - System calls (← I'll focus here)
  - Pseudo-file systems (/proc, /sys, etc.)
  - *ioctl()* interfaces (device drivers)
  - Netlink sockets
  - Obscure pieces (AUXV, VDSO, ...)

Interface designs must be done  
**right first time**



# Why must interfaces be right first time?

- Code changes  $\neq$  API changes



# Why is fixing interfaces so hard?

- Usually, “fixing” an interface means *breaking* the **ABI** for existing binaries





# Why is fixing interfaces so hard?

- Usually, “fixing” an interface means **breaking** the **ABI** for existing binaries



*“Dammit, I'm continually surprised by the \*idiots\* out there that don't understand that binary compatibility is one of the absolute top priorities. **The \*only\* reason for an OS kernel existing in the first place is to serve user-space. The kernel has no relevance on its own.**” [LKML, Mar 2012]*

***We have to live with our mistakes!***

***An interface design mistake by one  
kernel developer creates pain that  
thousands of user-space programmers  
must live with for decades***



# In truth, there are shades of gray

- 100% strict interpretation of ABI compatibility ==> **never** change ABI, not even to fix bugs (no matter how bad)
- In truth, ABI breakages are weighed against factors such as:
  - How important is it to make a bug fix?
  - How likely is existing user space to be affected by ABI change, and how much pain will be caused?
    - e.g., if interface is new, has very specialized uses, or has very few users, then maybe it can be changed



**So, what does it mean  
to get an API right?**



# Doing (kernel—user-space) APIs right

- Properly designed and implemented API should:
  - be bug free!
  - be as simple as possible (but no simpler)
  - be easy to use / difficult to misuse
  - be consistent with related/similar APIs
  - integrate well with existing APIs
    - e.g., interactions with `fork()`, `exec()`, threads, signals, FDs?
  - be as general as possible
  - be extensible, where needed; accommodate future growth trends
  - adhere to relevant standards (as far as possible) (e.g., POSIX)
  - be as good as, or better than, earlier APIs with similar functionality
  - be maintainable over time (a multilayered question)



**So how do kernel  
developers score?**



# Bugs



# Bugs

- *utimensat(2)* [2.6.22]
  - Set file timestamps
  - Multiple bugs!
    - [http://linux-man-pages.blogspot.com/2008/06/whats-wrong-with-kernel-userland\\_30.html](http://linux-man-pages.blogspot.com/2008/06/whats-wrong-with-kernel-userland_30.html)
  - Fixed in 2.6.26
- *signalfd()* [2.6.22]
  - Receive signals via a file descriptor
  - Didn't correctly obtain data sent with *sigqueue(2)*
  - Fixed in 2.6.25





# Bugs

- Examples of other interfaces with significant, easy to find bugs at release:
  - *inotify* [2.6.13]
  - *splice()* [2.6.17]
    - <http://marc.info/?l=linux-mm&m=114238448331607&w=2>
  - *timerfd* [2.6.22]
    - <http://marc.info/?l=linux-kernel&m=118517213626087&w=2>

# Bugs—what's going on?

- There's a quality control issue; **way too many** bugs in released interfaces
- Pre-release testing insufficient and haphazard:
  - Too few testers (maybe just kernel developer)
  - No unit tests
  - Insufficient test coverage
  - No clear specification against which to test
- Even if bug is fixed, users may still need to care
  - special casing for kernel versions



# Thinking about design



# Code it now, think about it later

- Vanishing arguments:
  - *readdir(2)* ignores *count*
  - *getcpu(2)* [2.6.19] ignores *tcache*
  - *epoll\_create()* [2.6] ignores *size* (must be  $> 0$ ) since 2.6.8
- Probably, argument wasn't needed to start with
  - Later recognized as a bad idea and made a no-op



# Code it now, think about it later

- *futimesat()* [2.6.16]
  - Extends *utimes()*
  - **Proposed** for POSIX.1-2008
  - Implemented on Linux
  - POSIX.1 committee realizes API is insufficient  
→ standardizes different API
  - *utimensat()* added in Linux 2.6.22



# Code it now, think about it later

- Dec 2003: Linux 2.6 added *epoll\_wait()*
  - File descriptor monitoring
    - (improves on *select()*)
  - Nov 2006, 2.6.19 added *epoll\_pwait()* to allow manipulation of signal mask during call
    - Superset of *epoll\_wait()*
- But, already in 2001, POSIX specified *pselect()* to fix analogous, well-known problem in *select()*



# Consistency



# Interface inconsistencies

- *mlock(start, length)*:
  - Round *start* down to page size
  - Round *length* up to next page boundary
  - *mlock(4000, 6000)* affects bytes 0..12287
    - (assuming page size is 4096 bytes)
- *remap\_file\_pages(start, length, ...)* [2.6]:
  - Round *start* down to page boundary
  - Round *length* **down** to page boundary(!)
  - *remap\_file\_pages(4000, 6000, ...)* ? → **0..4095**
- Users expect similar APIs to behave similarly





# Confusing the users

- Various system calls allow one process to change attributes of another process
  - e.g., *setpriority()*, *ioprio\_set()*, *migrate\_pages()*, *prlimit()*
- **Unprivileged** calls require credential matches:
  - Some combination of caller's UIDs/GIDs matches some combination of target's UIDs/GIDs



# Confusing the users

- But, much inconsistency; e.g.:
  - *setpriority()*:  $uid == t-uid \parallel uid == t-uid$
  - *ioprio\_set()*:  $uid == t-uid \parallel uid == t-uid$
  - *migrate\_pages()*:  $uid == t-uid \parallel uid == t-suid \parallel uid == t-uid \parallel uid == t-suid$
  - *prlimit()*:  $(uid == t-uid \&\& uid == t-uid \&\& uid == t-suid) \&\& (gid == t-gid \&\& gid == t-guid \&\& gid == t-sgid) \quad \color{red}{!!!!}$
- Inconsistency may confuse users into writing bugs
  - **Potentially, security-related bugs!**

• <http://linux-man-pages.blogspot.com/2010/11/system-call-credential-checking-tale-of.html>



# Generality



# Is the interface sufficiently general?

- 2.6.22 added *timerfd(ufd, flags, utimerspec)*
  - Create timer that notifies via a file descriptor
- But API didn't allow user to:
  - Retrieve previous value when setting new timer value
  - Do a “get” to retrieve time until next expiration
    - <http://marc.info/?l=linux-kernel&m=118517213626087&w=2>
    - <http://lwn.net/Articles/245533/>
- Older APIs (*[gs]etitimer()*, POSIX timers) **did** provide this functionality!

# Is the interface sufficiently general?

- Solution:
  - *timerfd()* disabled in kernel 2.6.23
  - 2.6.25 did it right:
    - *timerfd\_create()*, *timerfd\_settime()*, *timerfd\_gettime()*
    - (API analogous to POSIX timers)
- Was an ABI breakage, but
  - Only in a single kernel version
  - Original API was never exposed via *glibc*



# Are we learning from the past?



# Are we learning from past mistakes?

- Dnotify [2.4]
  - Directory change notification API
  - Many problems
- So, we added inotify [2.6.13]
  - *Much* better, but still has some problems
- Linux 2.6.37 added yet another related API, fanotify
  - Designed for virus scanners
  - Adds some functionality
  - Doesn't provide all functionality of inotify
- Couldn't we have had a new API that did everything?

# Extensibility





# Is the interface extensible?

- Too often, an early syscall didn't allow for extensions
- Common solution is *new* syscall, with *flags* arg:
  - *umount()* → *umount2()* [2.2]
  - *epoll\_create()* [2.6] → *epoll\_create2()* [2.6.27]
  - *futimesat()* [2.6.16] → *utimensat()* [2.6.22]
  - *signalfd()* [2.6.22] → *signalfd4()* [2.6.27]



# Is the interface extensible?

- When adding a new syscall, consider adding an (unused) *flags* argument to allow extensions
  - One success story: *finit\_module()* (Linux 3.8)
    - <http://thread.gmane.org/gmane.linux.kernel/1362095/focus=17490>



# Futureproofing

- Suppose a syscall has a *flags* bit-mask arg.
- Implementation should always have check like:

```
if (flags & ~(FL_X | FL_Y))  
    return -EINVAL;  
// Only allow caller to specify flags  
// bits that have a defined meaning
```

- Without this check, interface is “loose”



# Futureproofing

- Suppose user makes a call of form:

```
syscallxyz(-1); // flags has all bits set
```

- If implementer later adds **FL\_Z**, an ABI breakage occurs for user's code
- Conversely: user has no way of checking if a particular kernel implements **FL\_Z**
- Many system calls lack this kind of check!
  - Linux 3.2 examples: *sigaction(sa.sa\_flags)*, *recv()*, *send()*, *clock\_nanosleep()*, *msgrcv()*, *msgget()*, *semget()*, *shmget()*, *shmat()*, *semop(sops.sem\_flg)*



# Futureproofing

- Should checks be added after the fact?
  - e.g., `umount2()` [2.2] added check in 2.6.34;  
`timerfd_settime()` [2.6.25] added check in 2.6.29
- But adding check can also create ABI breakage
  - Apps get errors where previously they did not
    - e.g., kernel commit a8159414, `epoll_ctl()`, May 2012
      - <https://lwn.net/Articles/520198/>
- ***Loose APIs allow the user to define interface***
  - Worst case: can't add new *flags* values to interface



# Futureproofing failures

- 16 bits is enough for UIDs/GIDs...
  - 2.4: 32-bit UIDs/GIDs
- 32 bits is enough for file offsets
  - Okay, it was 1991, but Moore's law...
  - 2.4: 64-bit file offsets
- So we have
  - *oldstat()*, *stat()*, *stat64()*
  - *chown()*, *chown32()*
  - *open()*, *open64()*
  - and so on



# Maintainability



# When good ideas go astray

- Traditional UNIX gives *root* all privileges
  - All or nothing is risky!
- Linux capabilities divide *root* privileges into distinct pieces
  - Trade-off:
    - Want to split *root* into meaningfully separate pieces
    - Too many pieces becomes unmanageable





# When good ideas go astray

- Linux 3.2 has 36 capabilities:
  - 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\_SYSLOG, 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, CAP\_WAKE\_ALARM
- But which capability do I use for my new feature?
- I don't know... maybe CAP\_SYS\_ADMIN?
- CAP\_SYS\_ADMIN, the new root, 451 uses in 3.2
  - (out of 1167 total uses of CAP\_\*)
    - <https://lwn.net/Articles/486306/>

**Did we really mean to do  
that?**



# Did we really mean to do that?



*“Quite frankly, our most common ABI change is that we don't even realize that something changed. And then people may or may not notice it.” [LKML, Mar 2012]*

- *Many many examples of this...*

# RLIMIT\_CPU

- **RLIMIT\_CPU** resource limit governs amount of CPU time a process may use
- Soft limit (**SIGXCPU**) + hard limit (**SIGKILL**)
- Linux delivers **SIGXCPU** on hitting soft limit + each second thereafter until hard limit reached
- Linux 2.6.12: as well as invoking **SIGXCPU** handler, soft limit is raised by 1 sec. each time
- Behavior is surprising, nonstandard, nonportable, and unrequested
- Unlikely that this was an intended change



# Standards and portability



# Needlessly breaking portability

- *sched\_setscheduler()*
  - POSIX: successful call must return previous policy
  - Linux: successful call returns 0
  - **No** good reason for this inconsistency
  - Developers must special case code for Linux

**Actually,  
it wasn't just us...**



# We're just traditionalists...

- These kinds of problems predate Linux:
  - Using syscall function result to both return info on success and indicate an error creates problems
    - Some syscalls can return -1 on success (e.g., *getpriority()*)
  - API of System V IPC is awful!
  - Semantics of *fcntl()* locks when FD is closed render locks useless for libraries
  - *select()* modifies FD sets in place, forcing reinitialization inside loops
    - *poll()* gets it right: uses separate input and output args
  - and so on...





# Summary?

We could be doing a *lot*  
better at API design



# Why do these API problems keep happening?

- Excessive focus on code as primary contribution of value for a software project
- Poor feedback loop between developers and users



# Myth 2

Code is always the best way to contribute to Free Software



**“Show me the code!”**

*But anyone can write code,  
and if the design is good  
but the code is bad,  
the code can usually be fixed*



# “Show me the code!”

*Sometimes,  
other sentences are more appropriate,  
and encourage contributions that are  
as (or more) valuable*



**“Show me the users'  
requirements!”**



# “Show me the users' requirements”

- Does API serve needs of *multiple* users, or is it just one developer scratching an itch?
  - Beware of limited perspectives!
- Is API designed for **generality**?
- Is API **extensible** for possible future requirements?



**“Show me the design  
specification / documentation!”**





# “Show me the design spec. / documentation!”

- How do we know if implementation deviates from intention?
- What shall we code our tests against?
- Writing documentation turns out often to be a natural sanity check for design
- A decent man page suffices
  - Most of the bugs mentioned earlier were found while writing man pages...
  - Just a question of when man page is written...



**“Show me the design spec. / documentation!”**

*“Programming is not just an act of telling a computer what to do: it is also an act of telling other programmers what you wished the computer to do. Both are important, and the latter deserves care.”*

*[Andrew Morton, LKML, Mar 2012]*



**“Show me the design review!”**



# “Show me the design review!”

- Did other people actually review your design?
- Is API:
  - as simple as possible?
  - easy to use / difficult to misuse?
  - consistent with related/similar APIs?
  - well integrated with existing APIs?
  - as general as possible
  - extensible?
  - following standards, where relevant?
  - at least as good as earlier APIs with similar functionality?
  - maintainable?



**“Show me the tests!”**



# “Show me the tests!”

- Did you (the developer) write some tests?
- Did **someone else** write some tests?
- Do the tests cover all reasonable cases?
- Do you test for *unreasonable* cases?
  - Do unexpected inputs generate suitable error returns?
- While writing tests, did you find the interface easy to use / difficult to misuse?
  - (Did you consequently make some design changes?)



# Finally...

- *If you're a potential contributor, don't fall into the trap of believing that code is the only (or best) vehicle for contribution*
- *As a maintainer, are you encouraging these other types of contribution?*



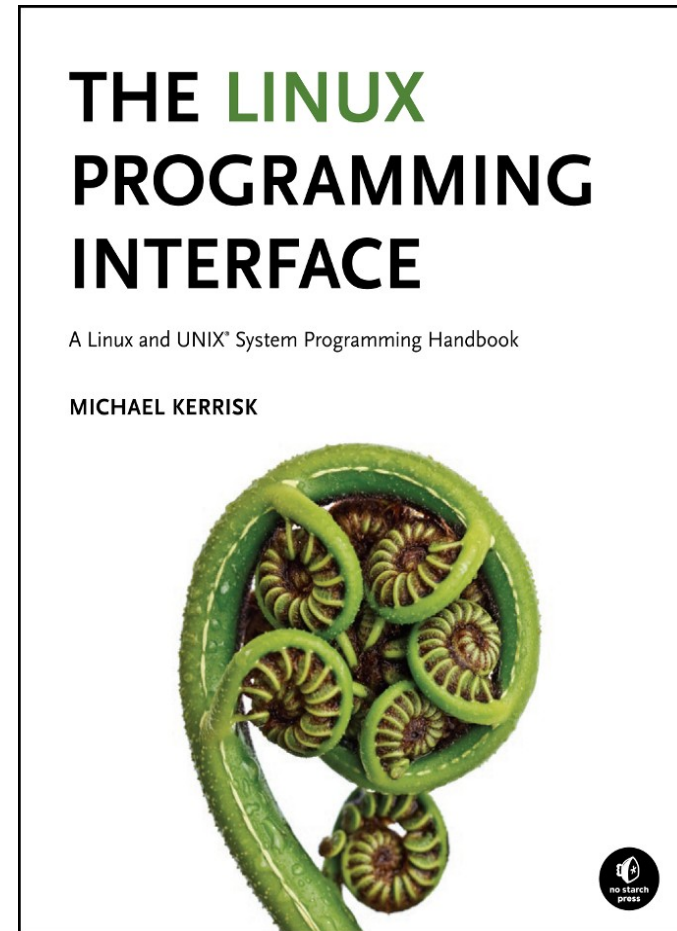
# Thanks! And Questions

(slides up soon at <http://man7.org/conf/>)

Michael Kerrisk  
[mtk@man7.org](mailto:mtk@man7.org)  
<http://man7.org/tlpi>

LWN.net  
[mtk@lwn.net](mailto:mtk@lwn.net)  
<http://lwn.net/>

Linux *man-pages* project  
[mtk.manpages@gmail.com](mailto:mtk.manpages@gmail.com)  
<http://www.kernel.org/doc/man-pages/>



Mamaku (Black Tree Fern) image (c) Rob Suisted  
[naturepic.com](http://naturepic.com)

(No Starch Press, 2010)