What we lose without words

Michael Kerrisk, Google Switzerland

LinuxConf Europe 2007
Cambridge, UK; 2 Sep. 2007

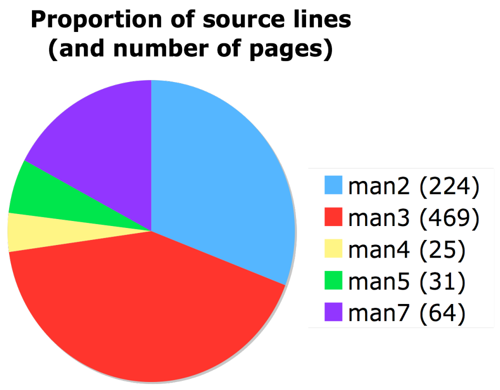
`www.kernel.org/pub/linux/docs/manpages`
`mtk-manpages@gmx.net`

The *man-pages* project

- Documents Linux kernel-userland API…
- and (GNU) C library API
- Sections 2, 3, 4, 5, and 7 of manual pages

Contents of *man-pages*

- As at *man-pages-2.64*:
  - ~800 man pages (== ~2000 printed pages)
  - 2: system calls
  - 3: library functions (*glibc*)
  - 4: devices
  - 5: file formats
  - 7: overviews, etc



What happens when we write documentation?

The Problems

- Many bugs in new interfaces released in stable kernels
- Some interface designs are poor
  - Arnd Bergmann, Monday 16.30: “How to not design kernel interfaces”
- All of the following examples were found while writing man pages

Buggy interfaces - 1

inotify

- File change notification API
- Appeared in kernel 2.6.13
- 2.6.16-rc timeframe, I wrote *inotify(7)*
- Testing: `IN_ONESHOT` had *never* worked
- Bug reported; fixed for 2.6.16

Buggy interfaces - 2

splice()

- transfer data between file descriptors without going through user space
- Appeared in kernel 2.6.17
- Simple test programs easily caused hangs (unkillable programs)
- Bug reported; fixed for 2.6.18

Buggy interfaces - 3

timerfd()

- New in kernel 2.6.22
- Create an interval timer that delivers expirations via a file descriptor
- *read()* returns 4-byte integer: count of expirations since last *read()*
- Bug: only least significant byte of count was returned

Buggy interfaces

- Many other bug found while writing man pages
  - (Some fixed during rc phase.)

Buggy interfaces - what’s the problem?

- These bugs should never have made it into stable kernels
- Insufficient testing during *rc* phase
- Testing is often *ad hoc*
  - Too few testers (often just kernel developer)
  - No unit tests
  - Insufficient test coverage

Buggy interfaces - why does documentation help?

- Documentation goes hand-in-hand with testing
- Documentation broadens audience who can understand interface and thus test it
- Documentation allows testers to determine if *implementation* == *intention*
- Good, early documentation → more & earlier testing → fewer released bugs

Interface design

- It's hard to design good programming interfaces
- Getting design wrong is painful…
  - Using interface is difficult, and bug-prone
  - APIs are forever; difficult/impossible to change design

Characteristics of a good interface

- Simplicity
- Ease of use
- Generality
- Consistency with similar interfaces
- Integration with related interfaces
- Extensible

When interface design goes wrong

dnotify (kernel 2.4; file change notification)

- Many problems in interface design
  - Uses signals (asynchronous notification)
  - granularity: only per directory (not single files)
  - requires open file descriptors (can’t unmount FS)
  - limited information provided in event notification
- Problems led to replacement by *inotify*
- But is the problem the developer(s)?
- Or the process?

Interface consistency: wrong (1)

- Two memory-related syscalls: *mlock()* and *remp_file_pages()*
- Both specify a *start* address + a *length*
- In *mlock(start, length)*:
  - Round *start* down to page size
  - Round *length* up to next page boundary
  - *mlock(4000, 6000)* affects bytes 0..12287

Interface consistency: wrong (2)

remap_file_pages(start, length, ...):

- Why settle just for inconsistent…
  - Round *start* down to page boundary
  - Round *length* **down** to page boundary(!)
- … when you can also have bizarre:
  - What address range is affected by *remap_file_pages(4000, 6000, ...)* ?

Interface problems with *timerfd()*

- Very useful; but provides less functionality than two previous timer interfaces:
  - *setitimer()/getitimer()*
  - *timer_create(), timer_settime(), timer_gettime()*
- These interfaces:
  - Provide a "get" interface to return time remaining until next timer expiration
  - Allow caller to retrieve time to next expiration when setting new timer value

Interface problems with *timerfd()* (2)

- *timerfd()* reinvents the wheel rather than leveraging existing API
- Arguably, a better design might have integrated with existing *timer_** functions:

```
timer_create(clockid, &evp, &timerid);
fd = timerfd(timerid);
// then use timer_settime() and
// timer_gettime() as normal
```

Interface design: what’s the problem?

- Insufficient review of interface designs
- Perhaps also: the implementers/designers of the interfaces (kernel developers) are often not the users (userland programmers)

Interface design: why documentation helps

- By its nature, writing documentation leads to self-review by designer(s)/implementer(s)
- Documentation broadens audience who can understand and critique design
- Existing documentation can be reviewed to look for consistency and integration of APIs

Dealing with a myth

"Documentation is fantasy: you have to read the source code to know the truth."

Problems with reading the source (1)

- The kernel is **big**:
  - 2.6.23 kernel source (`*.[chS]`) is **7.8M lines**
- and constantly changing:
  - Recent 2.6.x *diff –u* patches ~ **1M lines**
- Reading the code takes too much time
  - We need summaries of code: documentation

Problems with reading the source (2)

- If code doesn’t match documentation, which is right?
- What if implementation doesn’t match intention? (I.e., a bug)
- If no documentation, what defines the interface standard? Should it be the code?
- Forcing userland programmer to read code leads to a tightly constrained API

Proposal: formalize kernel-userland interface development (1)

- Goals
  - We should have fewer bugs in released interfaces
  - We need to do an excellent job of API design, because APIs are forever
- Apply formal sign-off requirements before new kernel-userland interfaces can be added:
  - API design review
  - Thorough documentation of interface
  - A full suite of userland test programs

Proposal: formalize kernel-userland interface development (2)

- Documentation should be written by/in collaboration with kernel developer
- At least some test code should be written by someone other than the developer
  - Too difficult for one person to consider all test cases

Before saying no…

- Consider that good documentation can help prevent:
  - Poorly designed/inconsistent interfaces
  - Bugs in new and changed interfaces
- Look at long list of FIXMEs in man pages;
- Note several system calls still lack man pages
- There are kernel coding standards; why not documentation (and testing) standards?

Helping with man-pages

- Testing new interfaces and helping document them is a great way of making a difference to the quality of the Linux API
  - Detect bugs earlier
  - Help improve API designs
  - You don't need to be a kernel developer
- Read `HOWTOHELP` in *man-pages* tarball at

`www.kernel.org/pub/linux/docs/manpages`

`mtk-manpages@gmx.net`

Michael Kerrisk, Google Switzerland

Thanks!

`www.kernel.org/pub/linux/docs/manpages`