blog.farhan.codes

Farhan's Personal and Professional Blog

Linux maintains bugs, The real reason ifconfig on Linux is deprecated

Posted on 2018-06-25 in unix • 919 words • 5 minute read
Tags: freebsd, C

In my third installment of FreeBSD vs Linux, I will discuss underlying reasons for why Linux moved away from ifconfig(8) to ip(8).

In the past, when people said, “Linux is a kernel, not an operating system”, I knew that was true but I always thought it was a rather pedantic criticism. Of course no one runs just the Linux kernel, you run a distribution of Linux. But after reviewing userland code, I understand the significant drawbacks to developing “just a kernel” in isolation from the rest of the system.

Lets say a userland program wants to request an object from the kernel. The kernel structure might be something like this:

```
struct foo {
     size_t size;
     char name[20];
     int val;
};
```

On POSIX systems, a typical way to communicate with the kernel is to open a file descriptor to the appropriate system and send an ioctl(1) with a pointer to where the kernel should store the responding data. FreeBSD might perform this task as follows:

```
struct foo x;
ioctl(fd, CMD_REQUEST_FOO, &x);
```

Linux should do the same and to be fair it typically does. This manifests as software source that requires the Linux kernels headers. But because userland tools are maintained independent of the kernel, and sometimes are even explicitly written to be cross-platform, they typically maintain their own copy of data structures and macros independent of the Linux source tree.

So far so good. This might even produce the exact same binary output. But what happens if the kernel structure or behavior changes? This could be due to a bug fix, an added feature or an optimization – either way, the structure may change.

On FreeBSD this is not a problem. They update the kernel and userland tools in tandem. In fact, because both the kernel and userland application are in the same source tree they can even share the same header files. For 3rd party userland applications, FreeBSD provides highly stable libraries that do all the kernel-interactions, such as lib80211(3) – its worth noting that OpenBSD and NetBSD do not have these libraries because the kernel interface itself is highly stable anyways. FreeBSD even provides a COMPAT layer in the rare cases that an older binary fails to run on modern versions of FreeBSD.

Conversely on Linux, because the kernel and the rest of the operating system are not developed in tandem, this means updating or fixing a kernel struct would almost guarantee to break a downstream application. The only to prevent this would be to conduct regular massively coordinated updates to system utilities when the kernel changes, and properly version applications for specific kernel releases. Quite a herculean endeavor. This also explains why systemtap, one of Linux's many answers to dtrace(1), does not work on Ubuntu.

Also, Linux can never have an equivalent of a lib80211(3) because there is no single standard library set. Even for the standard C library set, Linux has Glibc, uClibC, Dietlibc, Bionic and Musl. Rather than guessing the underlying C library implementation or falling into "dependency hell", applications default to the most low-level implementation or their requested functionality. Some tools, such as ifconfig(8), resort to just reading from the /proc filesystem.

Linux's solution to this problem was to create a policy of never breaking userland applications. This means userland interfaces to the Linux kernel never change under any circumstances, even if they malfunction and have known bugs. That is worth reiterating. Linux maintains known bugs – and actively refuses to fix them. In fact, if you attempt to fix them, Linus will curse at you, as manifest by this email.

And this leads back to the topic. Have you ever wondered why nearly every distribution deprecated ifconfig(8), a standard networking tool dating back to classic Unix? When Linux first implemented multiple IPv4 addresses on the same physical interface, it did so by cloning the interface in software and assigning each clone a unique IPv4 address. For example, eth0 could be cloned with eth0:1, eth0:2, etc. From a programmatic perspective, eth0 still only had one IPv4 address. As time passed and developers updated the kernel, it allowed users to assign multiple IPv4 addresses directly to the same interface., bypassing the need for cloning.

But Linux's API has not changed. It still only returns a single legacy IPv4 address per interface. An interface could have multiple IPv4 addresses but ifconfig(8) will still only report a single address. In other words, as it currently stands ifconfig(8) lies to you. I do not fully understand they did not just update ifconfig(8) – random IRC rumors say there was a failed attempt due to ifconfig(8)'s convoluted code-base. But for whatever reason, this led to the completely new tool ip(8).

By contrast, FreeBSD just updates their ifconfig(8) in tandem with any kernel updates and there were no problems. Simple.

This also explains why Linux has multiple tools for seemingly highly correlated network tasks. Rather than working together to create a consolidate tool, Linux has iw(8), iwconfig(8) and brctl(8), etc, whereas FreeBSD just has different drivers for its ifconfig(8) implementation. For the record, I think ip(8)'s syntax is cleaner than ifconfig(8)'s syntax, as the latter is a victim of IPv4 legacy syntax. If both tools worked just fine, it might be worth having ifconfig(8) for legacy scripts during a transitionary period, but making ip(8) the future. That would be perfectly fine, but it would be ideal if both tools just worked, rather than needing to abandon the tool because it is broken.

Written with love a laptop running OpenBSD 6.3.

Find me around the web:
GitHub | GitLab | Fediverse