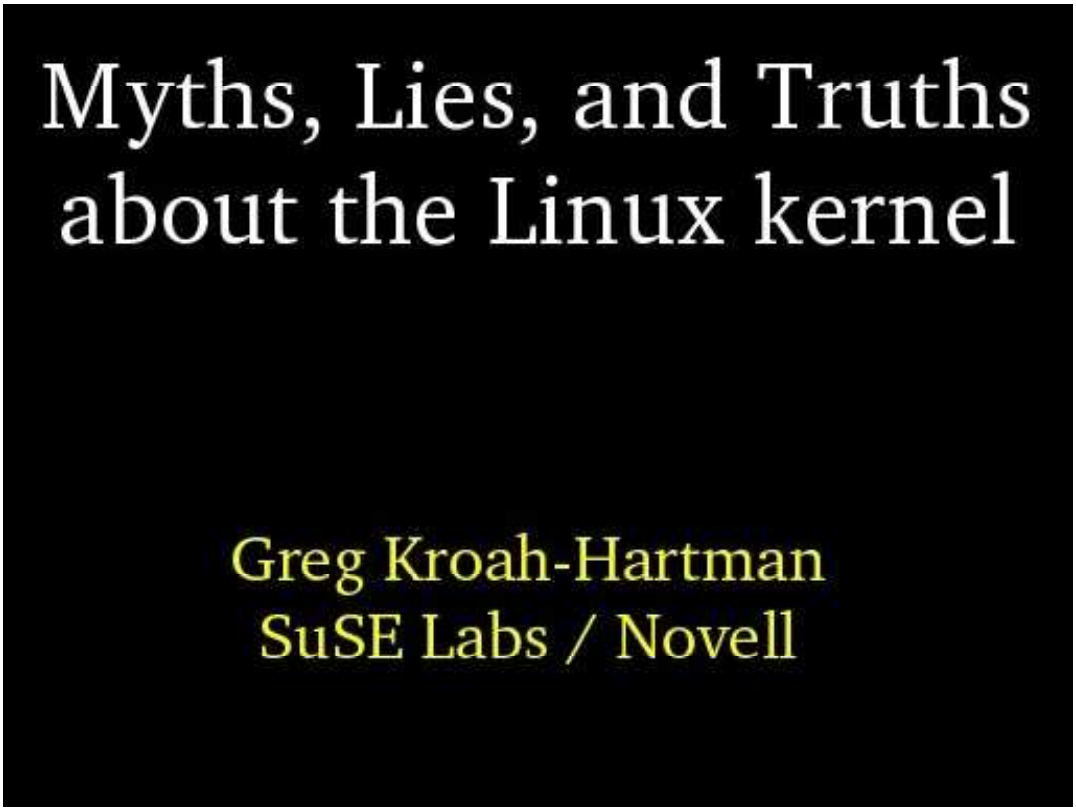
OLS 2006 Keynote

The organizers of OLS were kind enough to allow me the opportunity to speak this year as the closing keynote speaker. Here's the slides and text of my talk (well, the text is what I intended to say, the actual words that came out probably sounded a bit different.)

If you want to link directly to this talk, please use this link.

Myths, Lies, and Truths about the Linux kernel



Hi, as Dave said, I'm Greg, and I've been given the time by the people at OLS to talk to you for a bit about kernel stuff. I'm going to discuss the a number of different lies that people always say about the kernel and try to debunk them; go over a few truths that aren't commonly known, and discuss some myths that I hear repeated a lot.

Now when I mean a myth, I'm referring to something that was believed to have some truth to them, but when you really examine them, they are fictional. Let's call them the "urban myths" of the Linux kernel.

So, to start, let's look at a very common myth that really annoys me a lot:

Now I know that almost everyone involved in Linux has heard something like this in the past. About how Linux lacks device support, or really needs to support more hardware, or how we are lagging in the whole area of drivers. I've seen almost this same kind of quote from someone at OSDL a few months back, in my local paper, and it's extreme annoying.

This is really a myth, so people should really know better these days about saying this. So, who said this specific quote?:

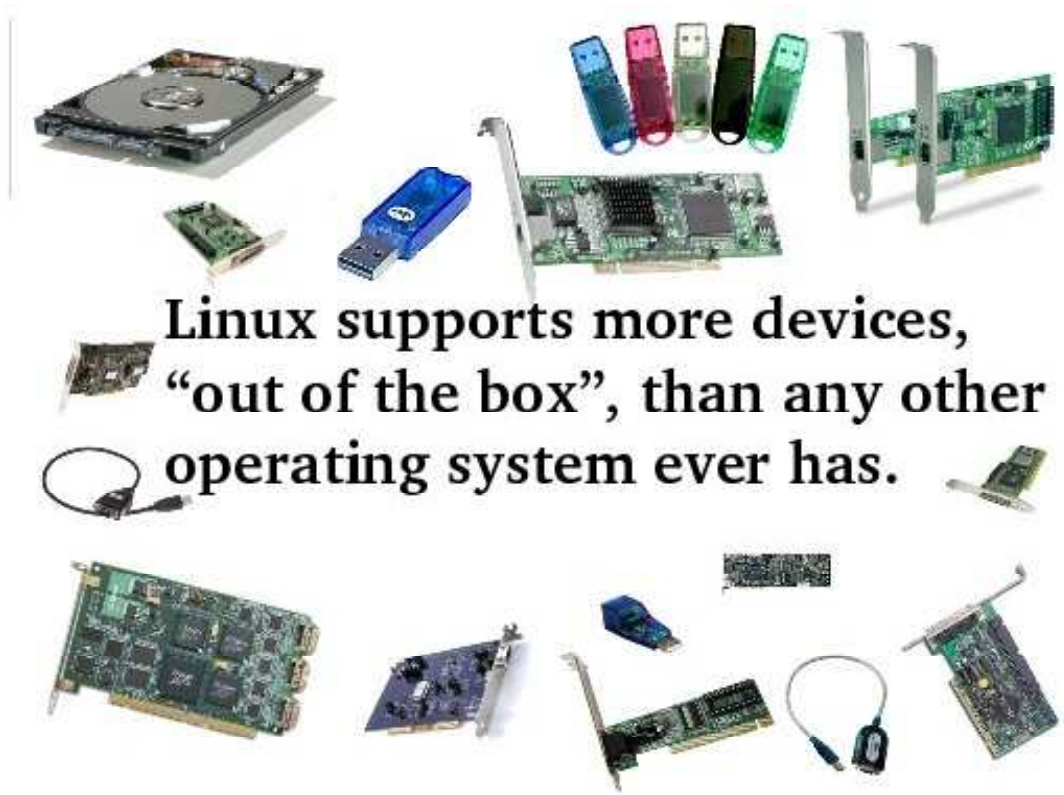
Ick.

Ok, well, he probably said this a long time ago, back when Linux really didn't support many different things, and when "Plug & Play" was a big deal with ISA buses and stuff:

Ugh.

Ok, so maybe I need to spend some time and really debunk this myth as it really isn't true anymore.

So, what is the fact concerning Linux and devices these days. It's this:



Yes, that's right, we support more things than anyone else. And more than anyone else ever has in the past. Linux has a very long list of things that we have supported before anyone else ever did. That includes such things as:

- USB 2.0
- Bluetooth
- PCI Hotplug

- CPU Hotplug
- memory Hotplug (ok, some of the older Unixes did support CPU and memory hotplug in the past, but no desktop OS still supports this.)
- wireless USB
- ExpressCard

and the list can go on, the embedded arena is especially full of drivers that no one else supports.

But there's a real big part of the whole hardware support issue that goes beyond just specific drivers, and that's this:

Yes, we passed the NetBSD people a few years ago in the number of different processor families and types that we support now. No other "major" operating system even comes remotely close in platform support for what we have in Linux. Linux now runs in everything from a cellphone, to a radio controlled helicopter, your desktop, a server on the internet, on up to a huge 73% of the TOP500 largest supercomputers in the world.

And remember, almost every different driver that we support, runs on every one of those different platforms. This is something that no one else has ever done in the history of computing. It's just amazing at how flexible and how powerful Linux is this way.

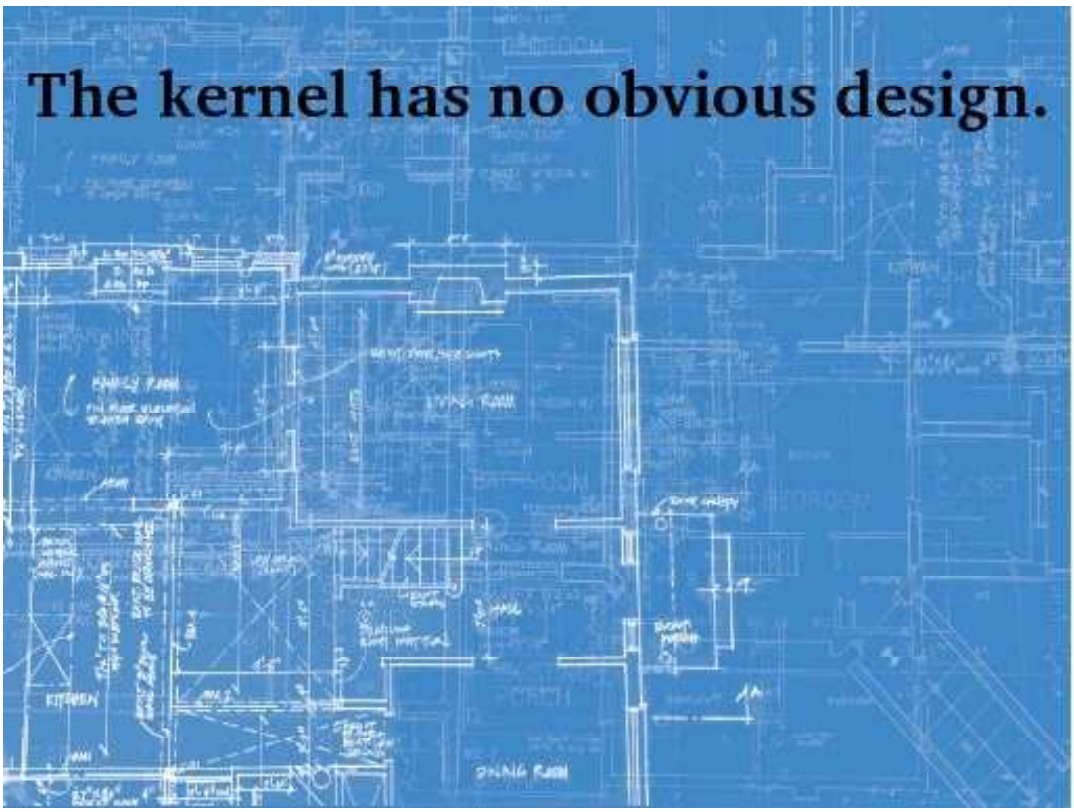
We now have the most scalable and most supported operating system that has ever been created. We have achieved something that is so unique and different and flexible that for people to keep repeating the "Linux doesn't support hardware" myth, is something that everyone needs to stop repeating. As it simply isn't true anymore.

Now, to be fair to Jeff Jaffe, when he said that original quote, he had just become the CTO of Novell, and didn't really have much recent experience with Linux, and did not realize the real state of device support that the modern distros now provide.

Look at the latest versions of Fedora, SuSE, Ubuntu and others. Installation is a complete breeze (way easier than any other operating system installation). You can now plug a new device in and the correct driver is automatically loaded, no need to hunt for a driver disk somewhere, and you are up and running with no need to even reboot.

An example of this, I recently plugged a new USB printer into my laptop, and a dialog box popped up and asked me if I wanted to print a test page on it. That's it, nothing else. If that isn't "plug and play", I really don't know what is.

But not everyone has been ignoring the success of Linux, as is obvious by the size of this conference. Lots of people see Linux and want to use it for their needs, but when they start looking deeper into the kernel, and how it is developed almost the first thing they run into is the total lack of a plan:



This lots of people absolutely crazy all the time. You see questions like "Linux has no roadmap so how can I create a product with it", and "How does anyone get anything done since no one is directing anyone", and other things like this.

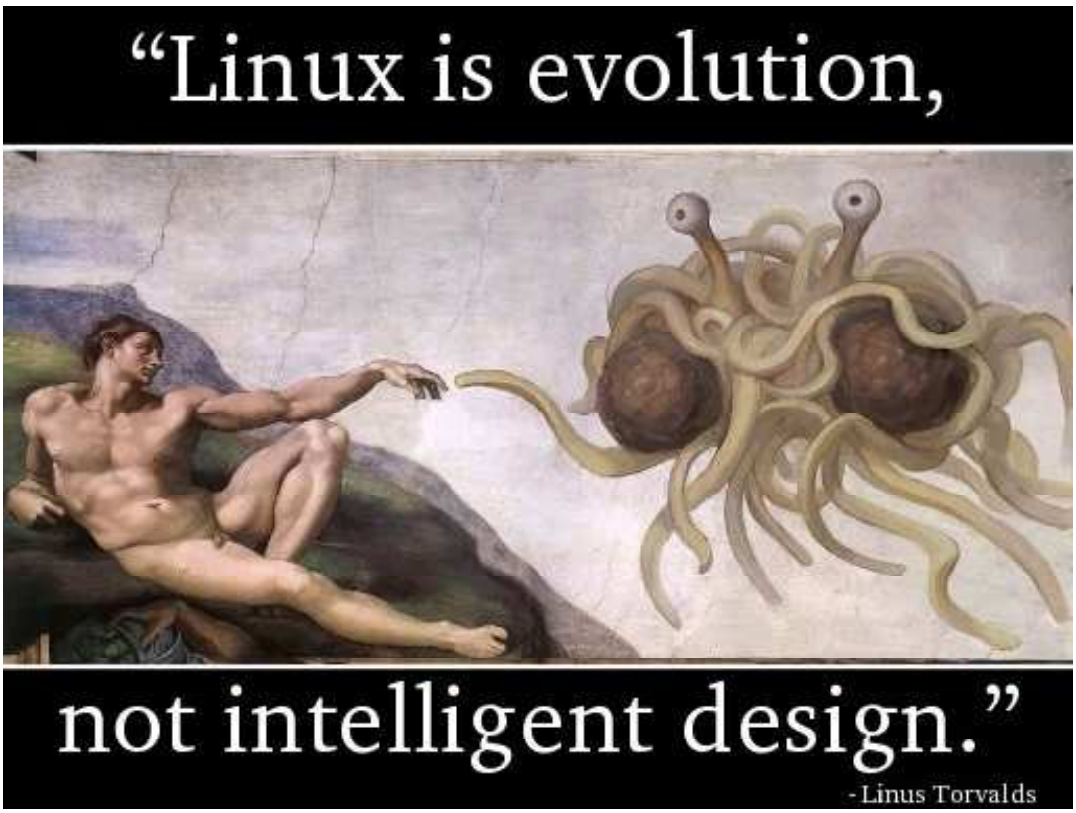
Well, obviously based on the fact that we are successful at doing something that's never been done before, we must have got here somehow, and be doing something right, but what is it?

Traditionally software is created by determining the requirements for it, writing up a big specification document, reviewing it and getting everyone to agree on it, implement the spec, test it, and so on. In college they teach software engineering methodology like the waterfall method, the iterative process method, formal proof methods, and others. Then there's the new ways of creating programs like extreme programming and top-down design, and so on.

So, what do we do here in the kernel?

Dr. Baba studies how businesses work and came to this conclusion after researching how the open source community works and specifically how the Linux kernel is developed and managed.

I guess it makes sense that since we have now created something that has never been done before, we did it by doing something different than anyone else. So, what is it? How is the kernel designed and created? Linus answered this question last year when he said the following to a group of companies when he was asked to explain the kernel design process:



This is a really important point that a lot of people don't seem to understand. Actually, I think they understand it, they just really don't like it.

The kernel is not developed with big design documents, feature requests and so on. It evolves over time based on the need at the moment for it. When it first started out, it only supported one type of processor, as that's all it needed to. Later, a second architecture was added, and then more and more as time went on. And each time we added a new architecture, the developers figured out only what was needed to support that specific architecture, and did the work for that. They didn't do the work in the very beginning to allow for the incredible flexibility of different processor types that we have now, as they didn't know what was going to be needed.

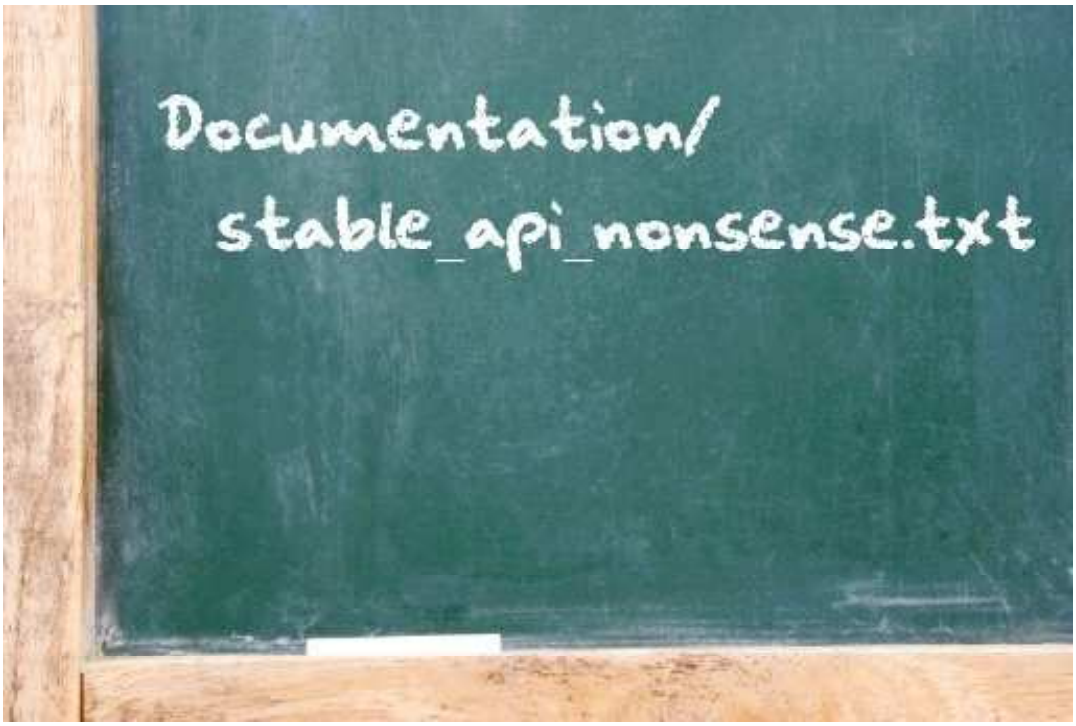
The kernel only changes when it needs to, in ways that it needs to change. It has been scaled down to tiny little processors when that need came about, and was scaled way up when other people wanted to do that. And every time that happened, the code was merged back into the tree to let everyone else benefit from the changes, as that's the license that the kernel is released under.

Jonathan on the first day of the conference showed you the huge rate of change that the kernel is under. Tons of new features are added at a gigantic rate, along with bug fixes and other things like cleanups. This shows how fast the kernel is still evolving, almost 15 years after it was created. It's morphed into this thing that is very adaptable and looks almost nothing like what it was even a few years ago. And that's the big reason why Linux is so successful, and why it will keep being successful. It's because we embrace change, and love it, and welcome it.

But one "problem" for a lot of people is that due to this constantly evolving state, the Linux kernel doesn't provide some things that "traditional" operating systems do. Things like an in-kernel stable API. Everyone has heard this one before:

For those of you who don't know what an API is, it is the description of how the kernel talks within itself to get things done. It describes things like what the specific functions are that are needed to do a specific task, and how those functions are called.

For Linux, we don't have a stable internal api, and for people to wish that we would have one is just foolish. Almost two years ago, the kernel developers sat down and wrote why Linux doesn't have an in-kernel stable API and published it within the kernel in the file:



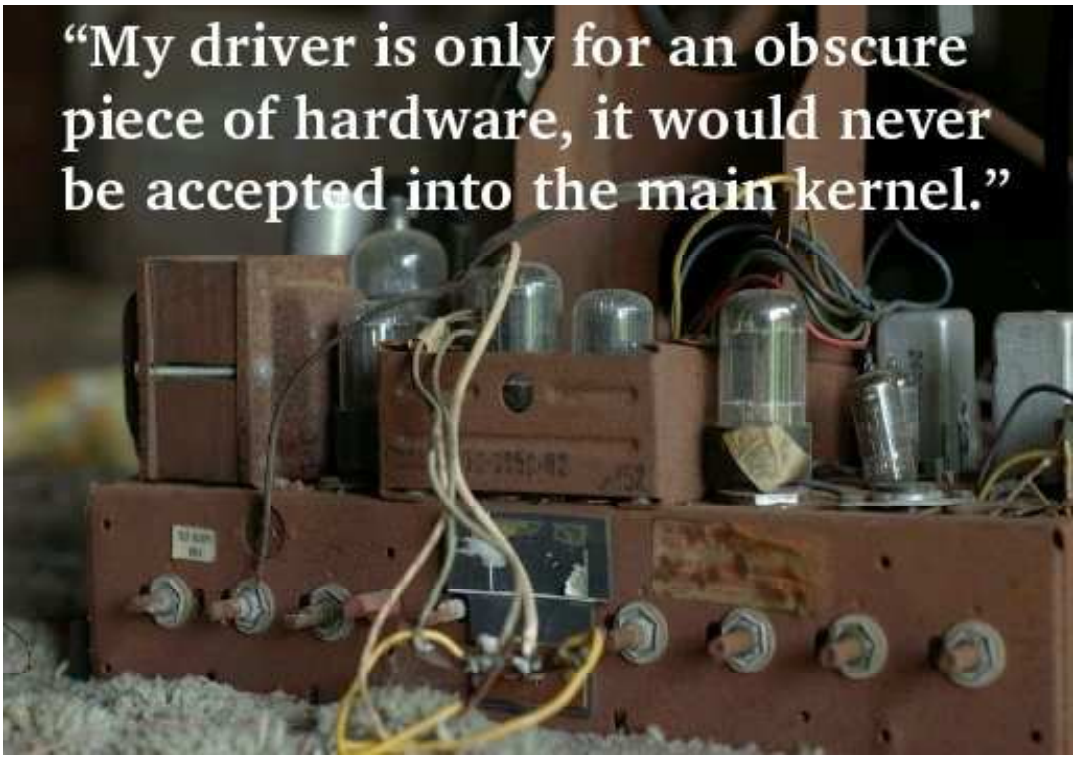
If you have any questions please go read this file. It explains why Linux doesn't have a stable in-kernel api, and why it never will. It all goes back to the evolution thing. If we were to freeze how the kernel works internally, we would not be able to evolve in ways that we need to do so.

Here's an example that shows how this all works. The Linux USB code has been rewritten at least three times. We've done this over time in order to handle things that we didn't originally need to handle, like high speed devices, and just because we learned the problems of our first design, and to fix bugs and security issues. Each time we made changes in our api, we updated all of the kernel drivers that used the apis, so nothing would break. And we deleted the old functions as they were no longer needed, and did things wrong. Because of this, Linux now has the fastest USB bus speeds when you test out all of the different operating systems. We max out the hardware as fast as it can go, and you can do this from simple userspace programs, no fancy kernel driver work is needed.

Now Windows has also rewritten their USB stack at least 3 times, with Vista, it might be 4 times, I haven't taken a look at it yet. But each time they did a rework, and added new functions and fixed up older ones, they had to keep the old api functions around, as they have taken the stance that they can not break backward compatibility due to their stable API viewpoint. They also don't have access to the code in all of the different drivers, so they can't fix them up. So now the Windows core has all 3 sets of API functions in it, as they can't delete things. That means they maintain the old functions, and have to keep them in memory all the time, and it takes up engineering time to handle all of this extra complexity. That's their business decision to do this, and that's fine, but with Linux, we didn't make that decision, and it helps us remain a lot smaller, more stable, and more secure.

And by secure, I really mean it. A lot of times a security problem will be found in one driver, or in one core part of the kernel, and the kernel developers fix it, and then go and fix it up in all other drivers that have the same problem. Then, when the fix is released, all users of all drivers are now secure. When other operating systems don't have all of the drivers in their tree, if they fix a security problem, it's up to the individual companies to update their drivers and fix the problem too. And that rarely happens. So people who buy the device, and then use the older driver that comes in the box with the device, which is insecure. This has happened a lot recently, and really shows how having a stable api can actually hurt end users, when the original goal was to help developers.

What usually happens after I talk to people about the instability of the kernel api, and how kernel development works, they usually respond with:

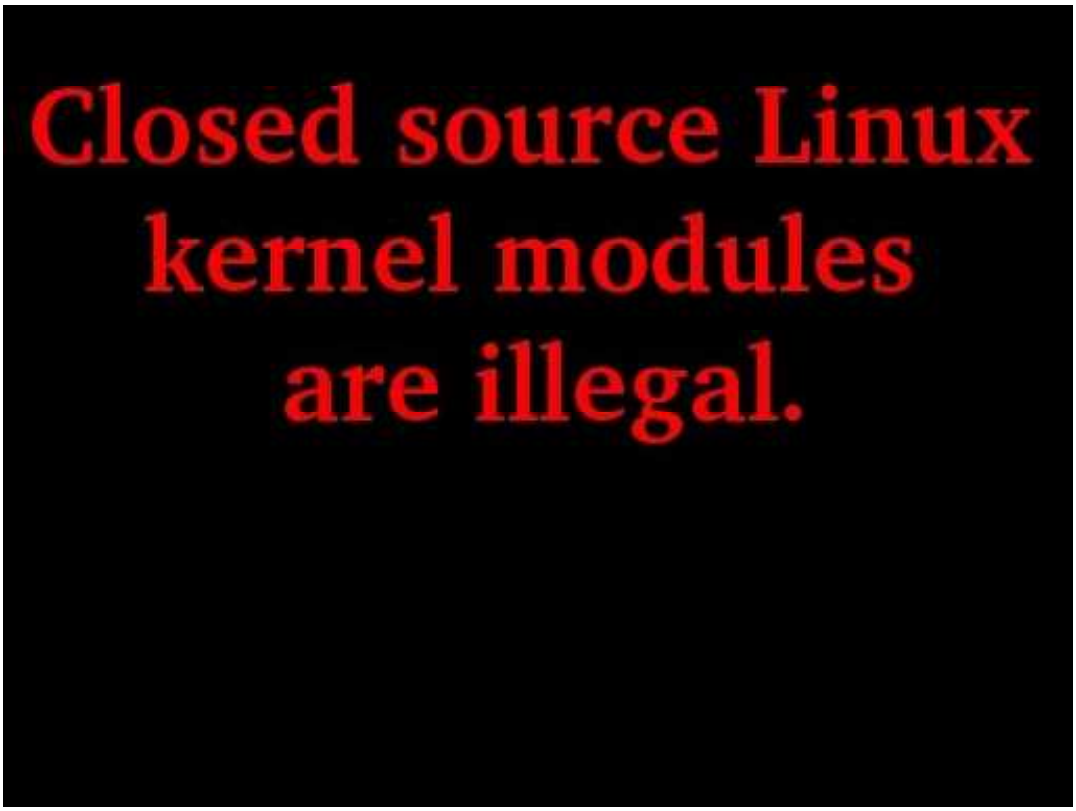


This just is not true at all. We have a whole sub-architecture that only has 2 users in the world out there. We have drivers that I know have only one user, as there was only one piece of hardware ever made for it. It just isn't true, we will take drivers for anything into our tree, as we really want it.

We want more drivers, no matter how "obscure", because it allows us to see patterns in the code, and realize how we could do things better. If we see a few drivers doing the same thing, we usually take that common code and move it into a shared piece of code, making the individual drivers smaller, and usually fixing things up nicer. We also have merged entire drivers together because they do almost the same thing. An example of this is a USB data acquisition driver that we have in the kernel. There are loads of different USB data acquisition devices out in the world, and one German company send me a driver a while ago to support their devices. It turns out that I was working on a separate driver for a different company that did much the same thing. So, we worked together and merged the two together, and we now have a smaller kernel. That one driver turned out to work for a few other company's devices too, so they simply had to add their device id to the driver and never had to write any new code to get full Linux support. The original German company is happy as their devices are fully supported, which is what their customers wanted, and all of the other companies are very happy, as they really didn't have to do any extra work at all. Everyone wins.

The second thing that people ask me about when it comes to getting code into the kernel is, well, we want to keep our code private, because it is proprietary.

So, here's the simple answer to this issue:



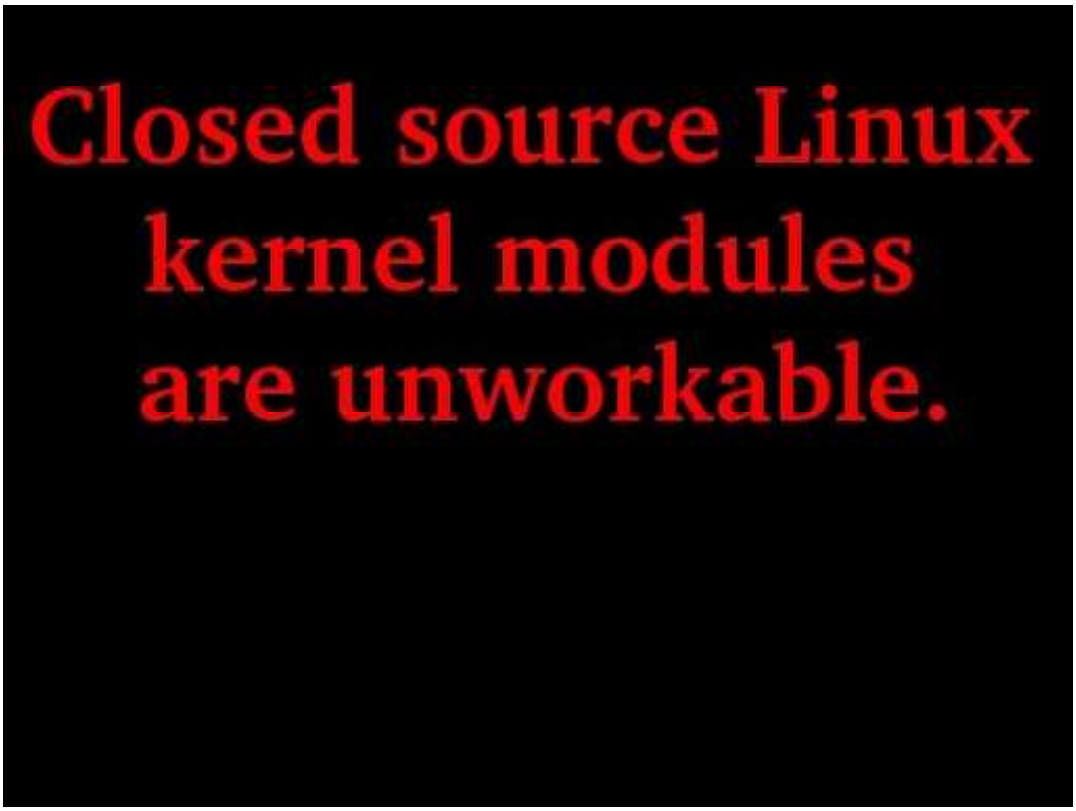
That's it, it is very simple. I've had the misfortune of talking to a lot of different IP lawyers over the years about this topic, and every one that I've talked to all agree that there is no way that anyone can create a Linux kernel module, today, that can be closed source. It just violates the GPL due to fun things like derivative works and linking and other stuff. Again, it's very simple.

Now no lawyer will ever come out in public and say this, as lawyer really aren't allowed to make public statements like this at all. But if you hire one, and talk to them in the client/lawyer setting, they will advise you of this issue.

I'm not a lawyer, nor do I want to be one, so don't ask me anything else about this, please. If you have legal questions about license issues, talk to a lawyer, never bring it up on a public mailing list like linux-kernel, which only has programmers. To ask programmers to give legal rulings, in public, is the same as asking us for medical advice. It doesn't make sense at all.

But what would happen if one day the Linux kernel developers suddenly decided to let closed source modules into the kernel? How would that affect how the kernel works and evolves over time?

It turns out that Arjan van de Ven has written up a great thought exercise detailing exactly what would happen if this came true:

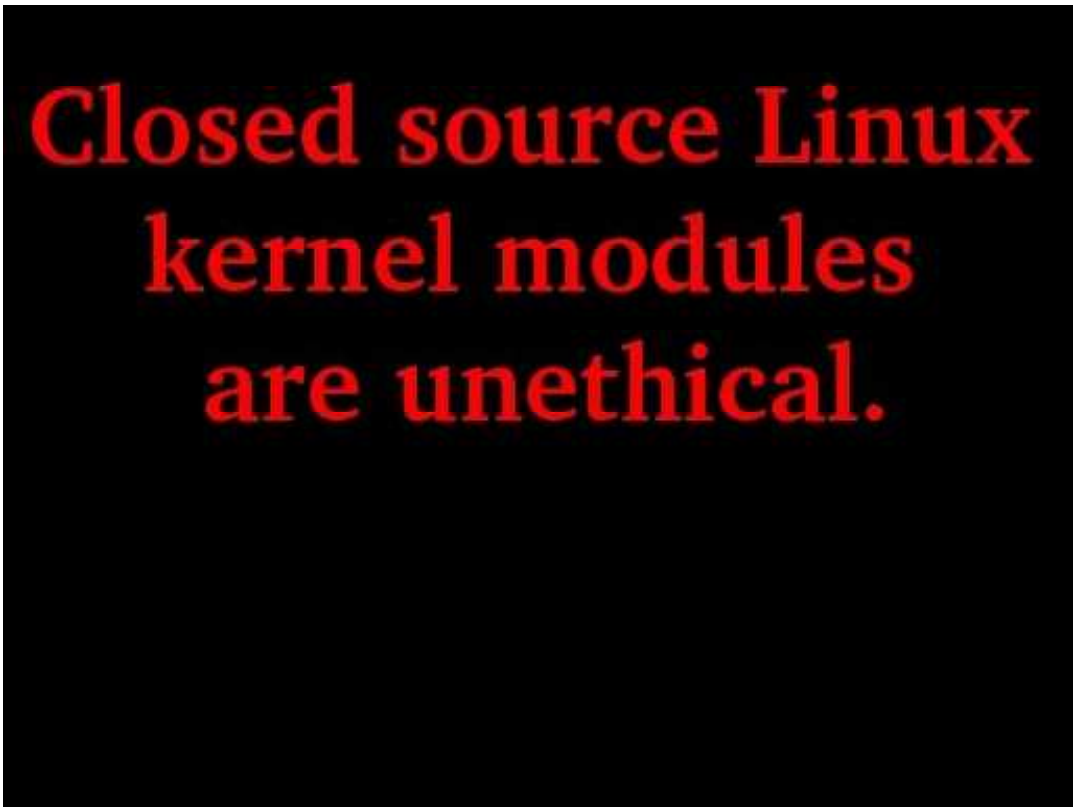


In his article, which can be found in the linux-kernel archives really easily, he described how only the big distros, Novell and Red Hat, would be able to support any new hardware that came out, but would slowly stagnate as they would not be allowed to change anything that might break the different closed source drivers. And then, if you loaded more than one closed source module, support for your system would pretty much be impossible. Even today, this is easily seen if you try to load more than one closed source module into your system, if anything goes wrong, no company will be willing to support your problem.

The article goes on to show how the community based distros, like Gentoo and Debian, would slowly become obsolete and not work on any new hardware platforms, and dry up as no users would be able to use them anymore. And eventually, in just a few short years, the whole kernel project itself would come to a standstill, unable to innovate or change anything.

It's a really chilling tale, and quite good, please go look it up if you are interested in this topic.

But there's one more aspect of the whole closed source module issue that I really want to bring up, and one that most people ignore. It's this:



Remember, no one forces anyone to use Linux. If you don't want to create a Linux kernel module, you don't have to. But if your customers are demanding it, and you decide to do it, you have to play by the rules of the kernel. It's that simple.

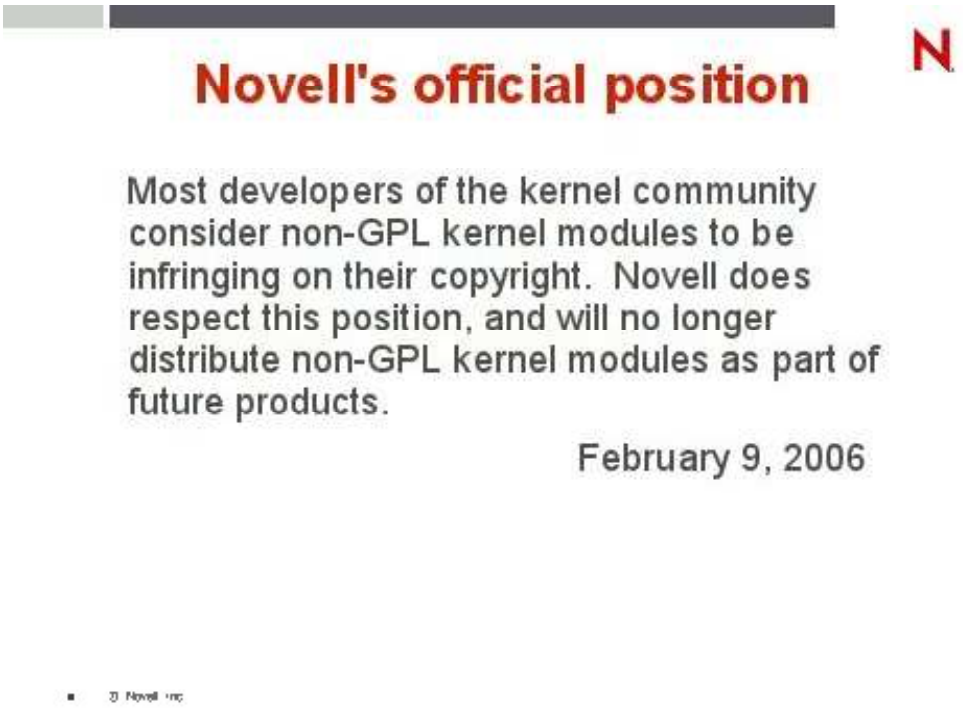
And the rule of the kernel is the GPL, it's a simple license, with standard copyright ownership issues, and many lawyers understand it.

When a company says that they need to "protect their intellectual property", that's fine, I and no other kernel developer has any objection to that. But by the same token, you need to respect the kernel developers intellectual property rights. We released our code under the GPL, which states in very specific form, exactly what your rights are when using this code. When you link other code into our body of code, you are obligated by the license of the kernel to also release your code under the same license (when you distribute it.)

When you take the Linux kernel code, and link or build with the header files against it, with your code, and not abide by the well documented license of our code, you are saying that for some reason your code is much more important than the entire rest of the kernel. In short, you are giving every kernel developer who has ever released their code the finger.

So remember, the individual companies are not more important than the kernel, for without the kernel development community, the companies would have no kernel to use at all. Andrew Morton stood up here two years ago and called companies who create closed source modules leaches. I completely agree. What they do is just totally unethical. Some companies try to skirt the license of the law on how they redistribute their closed source code, forcing the end user of it to do the building and linking, which then causes them to violate the GPL if they want to give that prebuilt module to anyone else. These companies are just plain unethical and wrong.

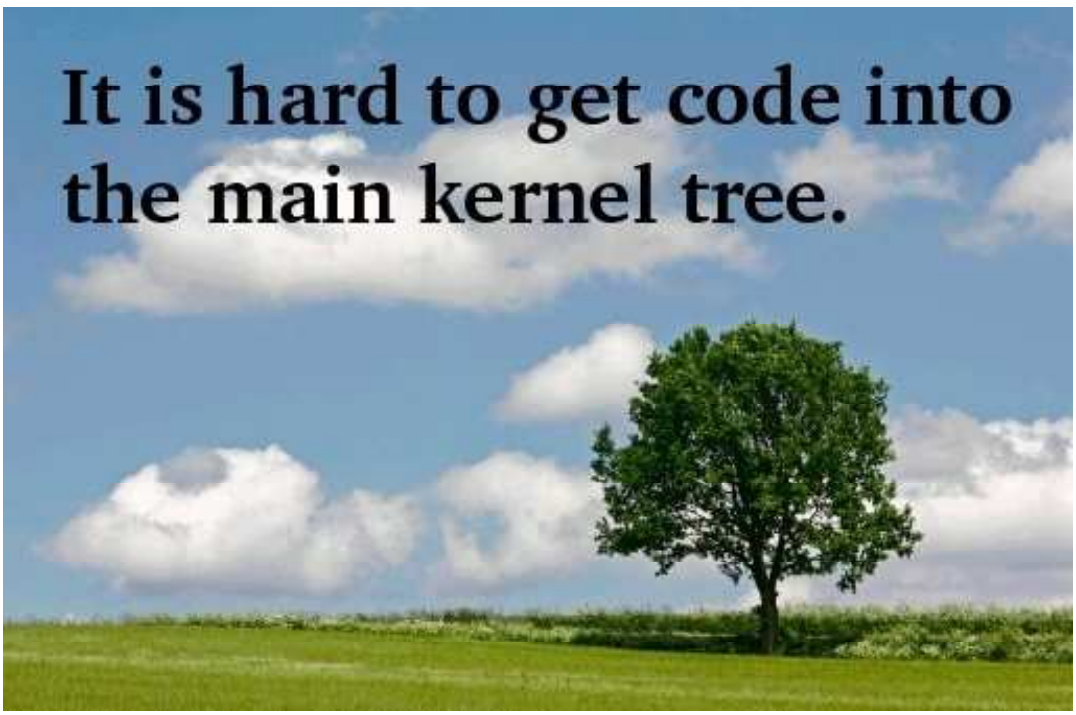
Luckily people are really starting to realize this and the big distros are not accepting this anymore. Here's what Novell publicly stated earlier this year:



This means that SuSE 10.1, and SLES and SLED 10 will not have any closed source kernel modules in it at all. This is a very good thing.

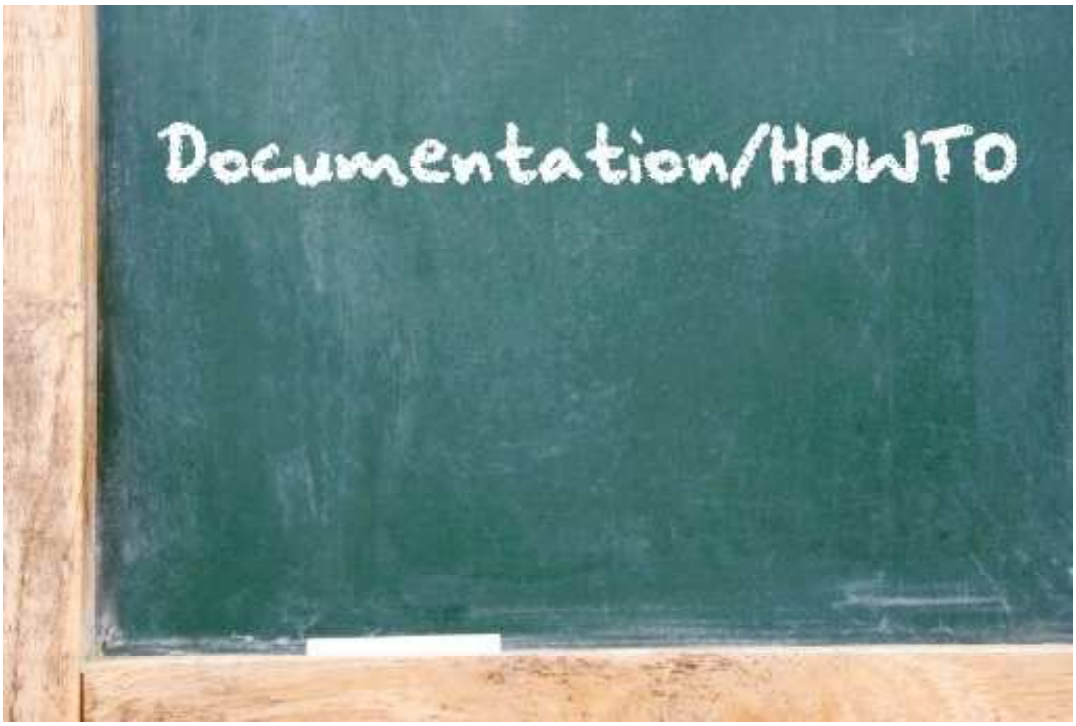
Red Hat also includes some text like this in their kernel package, but hasn't come out and said such a public statement.

Alright, enough depressing stuff. After companies realize that the really need to get their code into the kernel tree, they quickly run into one big problem:



This really isn't as tough of a problem as it first looks. Remember, the rate of change is about 6000 different patches a kernel release, so some one is getting their code into the tree.

So, how to do it. Luckily, the kernel developers have written down everything that you need to know for how to do kernel development. It's all in one file:



Please, point this file out to anyone who has questions on how to do kernel development. It answers about everything that anyone has ever asked, and points people at other places where the answers can be found.

It talks about how the kernel is developed, how to create a patch, how to find your way around the kernel tree, who to send patches to, what the different kernel trees are all about, and it even lists things you should never say on the linux-kernel mailing list if you expect people to take your code seriously.

It's a great file, and if you ever have anything that it doesn't help you out with, please let the author of that file know and they will work to add it. It should be the thing that you give to any manager or developer if they want to learn more about how to get their code into the kernel tree.

One thing that the HOWTO file describes, is various communities that can help people out with kernel development. If you are new to kernel development, there is the:

project. This is a very good wiki, a very nice and tame mailing list where you can ask basic questions without feeling bad, and there's also an IRC channel where you can ask questions in realtime to a lot of different kernel developers. If you are just starting out, please go here, it's a very good place to learn.

If you really want to start out doing kernel development, but don't know what to do, the:

project is an excellent place to start. They keep a long list of different "janitorial" tasks that the kernel developers have said it would be good to have done to the code base. You can pick from them, and learn the basics of how to create a patch, how to fix your email client to send a proper patch, and then, you get to see your name in the kernel changelog when your patches go in.

I really recommend this project for anyone who wants to start kernel development, but hasn't found anything specific to work on yet. It gets you to search around the kernel tree, fixing up odd things, and by doing that, you will usually find something that interests you that no one else is doing, and you can slowly start to take that portion of the kernel over. I can't recommend this group enough.

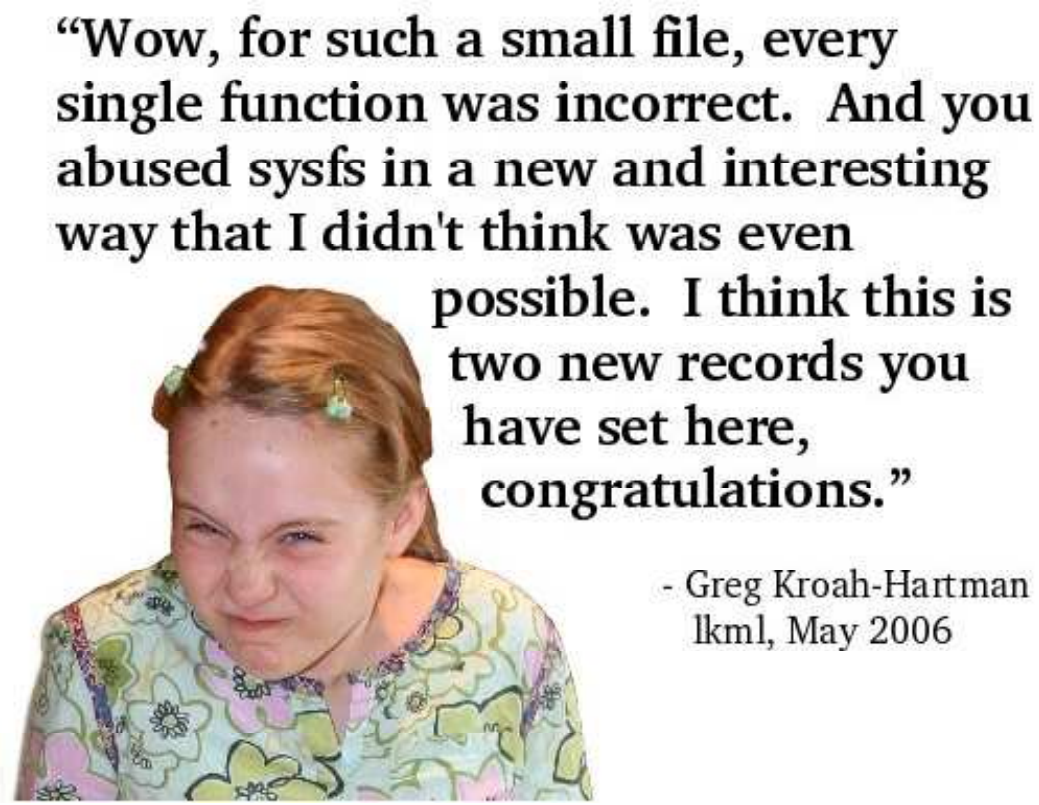
And then, there's the big huge mailing list, where everyone lives on:

This list gets about 200 emails a day to it, and can be hugely daunting to anyone trying to read it. Here's a hint, almost no one, except Andrew Morton, reads all of the emails on it. The rest of us just use filters, and read the things that interest them. I really suggest just finding some developers that you know provide interesting

commentary, and reading the threads they respond to. Or just search for subjects that look interesting. But don't try to read everything, you'll just never get any other work done if you do that.

The Linux kernel mailing list also has another kind of perceived problem. Lots of people can find the reaction of developers on this list as very "harsh" at times. They post their code, and get back scathing reviews of everything they did wrong. Usually the reviewers only criticize the code itself, but for most people, this can be a very hard thing to be on the receiving end of. They just put out what they felt was a perfect thing, only to see it cut into a zillion tiny pieces.

The big problem of this, is we really only have a very small group of people reviewing code in the kernel community. Reviewing code is a hard, unrewarding, tough thing to do. It really makes you grumpy and rude in a very short period of time. I tried it out for a whole week, and at the end of it, I was writing emails like this one:



Other people who review code, aren't even as nice as I was here.

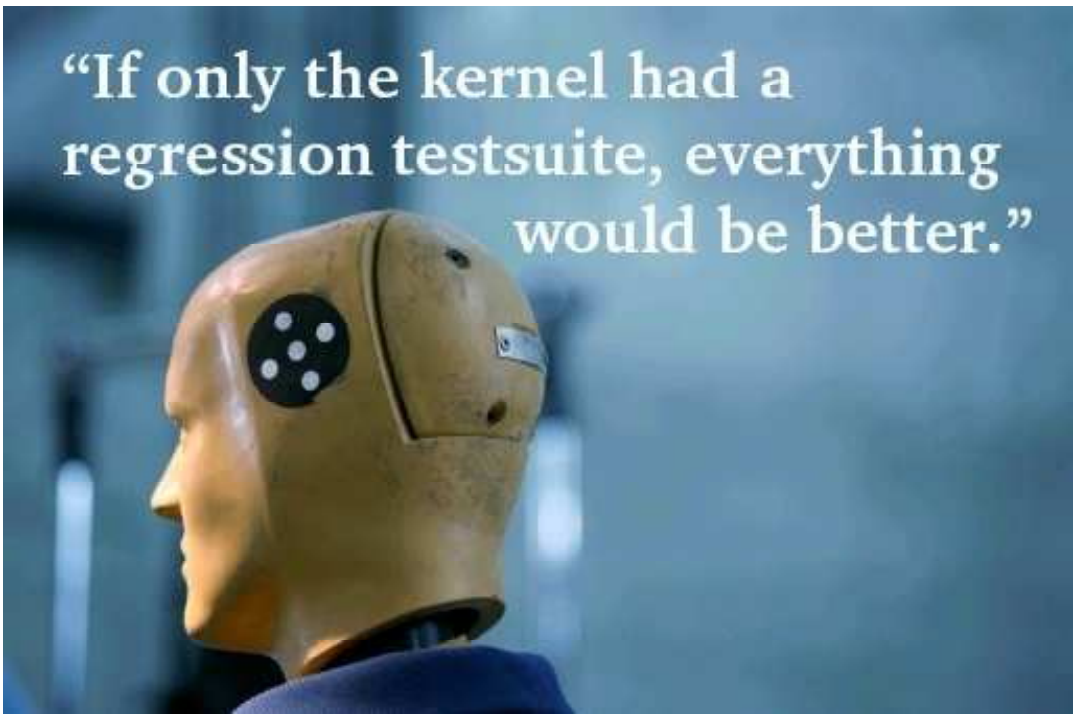
I'd like to publicly thank Christoph Hellwig and Randy Dunlap. Both of them spend a lot of time reviewing code on the linux-kernel mailing list, and Christoph especially has a very bad reputation for it. Bad in that people don't like his reviews. But the other kernel developers really do, because he is right. If he tells you something is wrong, and you need to fix it, do it. Don't ignore advice, because everyone else is watching to see if you really do fix up your code as asked to. We need more Christophs in the kernel community.

If everyone could take a few hours a week and review the different patches sent to the mailing list, it would be a great thing. Even if you don't feel like you are a very good developer, read other people's code and ask questions about it. If they can't defend their design and code, then there's something really wrong.

It's also a great way to learn more about programming and the kernel. When you are learning to play an instrument, you don't start out writing full symphonies on your own, you spend years reading other peoples scores, and learning how things are put together and work and interact. Only later do you start writing your own music, small tunes, and then, if you want, working up to bigger pieces. The same goes for programming. You can learn a lot from reading and understanding other people's code. Study the things posted, and ask why things are done specific ways, and point out problems that you have noticed. It's a task that the kernel really needs help with right now.

(possible side story about the quote)

Alright, but what if you want to help out with the kernel, but you aren't a programmer. What can you do? Last year Dave Jones told everyone that the kernel was going to pieces, with loads of bugs being found and no end in sight. A number of people made the response:



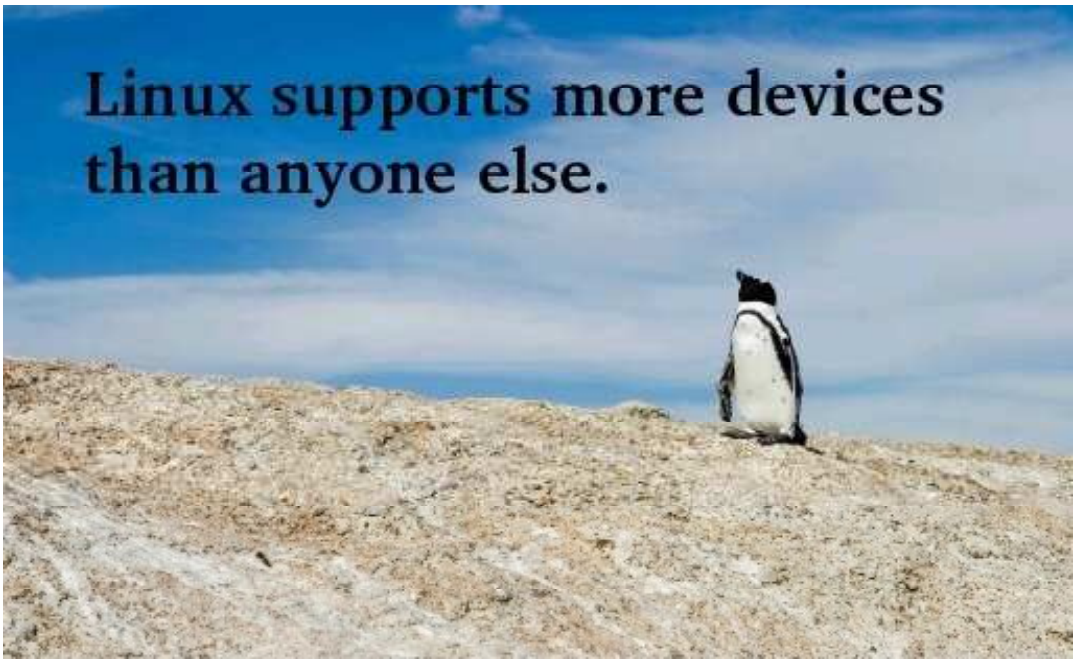
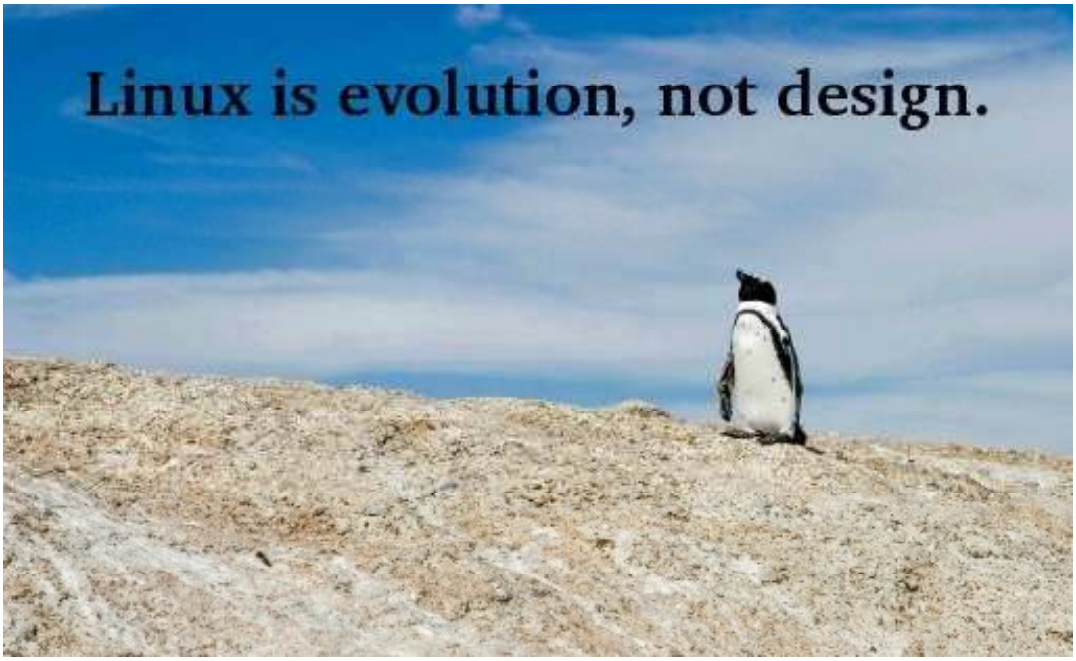
Now, this is true, it would be great to have a simple set of tests that everyone could run for every release to ensure that nothing was broken and that everything's just right. But unfortunately, we don't have such a test suite just yet. The only set of real tests we have, is for everyone to run the kernel on their machines, and to let us know if it works for them.

So, that's what I suggest for people who want to help out, yet are not programmers. Please, run the nightly snapshots from Linus's kernel tree on your machine, and complain loudly if something breaks. If no one pays attention, complain again. Be really persistent. File bugs in:

People do track things there. Sometimes it doesn't feel like it, but again, be persistent. If someone keeps complaining about something, we do feel bad, and work to try to fix

things. Don't feel bad about being a pest, because we need more pests to keep all of us kernel developers in line.

And if you really feel brave, please, run Andrew Morton's -mm kernel tree. It contains all of the different kernel maintainer's development trees combined into one big mass of instability. It is the proving ground for what will eventually go into Linus's kernel tree. So we need people testing this kernel out to report problems early, before they go into Linus's tree.

I wouldn't recommend running Andrew's kernels on a machine with data that you care about, that would not be wise. So if you have a spare machine, or you have a very good backup policy, please, run his kernels and let us know if you have any problems with stuff.

So finally, in conclusion, here's the main things that I hope people remember: