Kernel Traffic #83 For 5 Sep

By Zack Brown

Table Of Contents

Introduction

Thanks go to Tim Bell for catching some typos in the Timezone story last week. (That's the last time I spell "Australian Capital Territory" as "Au**a**tralian Capit**o**l Territory". Thanks Tim ;-) !

Thanks also go to Rasmus Andersen for catching me referring to Rik van Riel as Theodore Y. Ts'o -- Uh... don't know quite how that happened, but thanks Rasmus!

Thanks also go to Richard Hirst, Rob Landley, Alistair Lambie, and Mark Jefferys for pointing out that one of the back-issue links gave a 404 (I forgot to do a 'cvs add' on that file -- oops!). Thanks, folks!

Thanks also go to Tobias Diedrich, Jamie Fifield, Jochen Striepe and Ulrich Mayer for adding some timezone information to the the table from last issue. Thanks!

Chris Litchfield sent me a criticism of linux-kernel itself. He said:

> **One of the things that bothers me is the feeling of eliteness that Linux people have. Alan does not :) at least IMHO. He has replied to several programming questions across several platforms. However, the LACK of a common path for basic questions disturbs me.**
>
> **People send email to Kernel for not know where else to send it. They get flamed. It then turns off others from sending email and learning things. Many people can be turned off from the Process this way.**

I think this must be a fairly common conception of linux-kernel, and it's sad. But linux-kernel is just not the place to post if you don't know where to post. Folks who don't know what they're doing should go elsewhere. linux-kernel is a high-traffic kernel hacking list, not a newbie list. If newcomers to Linux (or anyone else, for that matter) post off-topic questions there, they may well get flamed, and for good reason.

On the other hand, it's not exaggerating to say that every single kernel developer wants and welcomes bug reports to linux-kernel from anyone in the world. linux-kernel is not a

"developers-only" list, where a core group labors in isolation. Those days are gone, and have been for years. linux-kernel is a place where ordinary users should feel OK to post their own bug reports. Every once in awhile, in an effort to lower the amount of traffic on the list, someone will suggest only allowing posts from list subscribers. But this is always rejected in the end, because the developers recognize the need to ensure that bug-reporting remains as easy a process as possible for anyone who might have a contribution to make.

Anyone with a problem that might be kernel related, *should* follow the guidelines in the file 'REPORTING-BUGS' in the Linux source directory, which should already be on your system if you're compiling your own kernels. If you follow those guidelines you won't get flamed, you might get your question answered, and you might end up making the kernel better.

Mailing List Stats For This Week

We looked at 855 posts in 3889K.

There were 322 different contributors. 142 (44%) posted more than once. 69 (21%) posted last week too.

The top posters of the week were:

- 41 posts in 200K by Arnaldo Carvalho de Melo <acme@conectiva.com.br>
- 40 posts in 178K by Tigran Aivazian <tigran@veritas.com>
- 26 posts in 109K by Linus Torvalds <torvalds@transmeta.com>
- 21 posts in 68K by Alan Cox <alan@lxorguk.ukuu.org.uk>
- 15 posts in 54K by Ingo Molnar <mingo@elte.hu>

1. 2.4 SMP Scalability

18 Aug - 22 Aug (11 posts): 2.4.x SMP scalability question.

A nice summary of where we are with SMP scalability, and the difficulties involved in making those assessments. Mike A.

Harris [*] asked with care:

> **With the 2.4 kernel nearing release, what is the perceived level of scalability that can usefully be achieved now? How many processors until you max out performance? If it varies from architecture to architecture, please indicate the differences.**
>
> **Please feel free to point me to URL's discussing this. I'm in a SMP scalability discussion of sorts, and people believe Linux can't usefully go past 4 processors. I figured that was true for 2.2.x but believe 2.4.x scales much further and just want to share the facts - whatever they may be - with people.**

Rik van Riel [*] pointed out that this was entirely dependent on the work done by the system. As he said, if an application weren't scalable, it wouldn't make use of SMP. And Ralf Baechle [*] reiterated, **"without knowing your particular load pattern an anwer can't be given."**

Ingo Molnar [*] also replied to Mike with some practical numbers, **"the 2.4 kernel's scalability of the most common workloads is limited by hardware (ie. the kernel lets those workloads scale 'down to the metal'). The 2.4 kernel scales quite decently on 8 CPUs. I would be surprised if we had any serious problem at 32 or 64 CPUs. (Linux right now is architecturally limited to 32 CPUs on 32-bit systems and 64 CPUs on 64-bit systems - because we have some per-CPU data structures put into word-size bitfields.) But fortunately there is still scalability work to be done - a perfect kernel would be extremely boring! :-)"**

Ralf replied that he, Kanoj Sarcar [*], Cort Dougan [*] and David S. Miller [*] had recently agreed to tackle the word-size-bitfield problem for 2.4, since 2.6 or 3.0 would be too far in the future considering the importance of the problem. Ingo replied that he'd protested word-sized bitfields when they were first introduced, and added now, **"Basically i think it's not hard at all to avoid word-sized bitmasks (or to define long enough ones) - if it's done right from the start."**

Matthew Wilcox [*] also replied to Ingo's assessment of 8-CPU scalability, arguing, **"The SpecWeb numbers seemed to show that Linux didn't scale nearly as well to 8 CPUs as it did to 4. Or was that the SpecWeb benchmark not scaling well to 8 CPUs? (From memory, we were acheiving 80%+ scalability on 2 and 4 CPUs and then dropped to 65% on the 8 CPU test -- equivalent to using 5 CPUs at 100%.)"** But Ingo replied, **"the SpecWeb benchmark is not something that is scaled linearly easily. It has a quadratically increasing memory footprint, and it has a workload component that forces heavy memory traffic."** He added, **"i can show you pure user-space workloads that will not scale past 2 CPUs on any x86 SMP system."**

In reply to this, someone asked about how to diagnose the different SMP benchmarks, and Alan Cox [*] explained, **"The critical one on a PC is almost always memory bandwidth. Running a tight loop on one CPU doing chained dereferences to the same cache colour (ie continually cycling lines through cache) will heavily dent performance of the whole machine. You really do have to try hard to get x86 apps on heavily smp scaled boxes running from cache."**

It would be interesting to hear about the difficulties of coding for things like SMP, when it's so difficult to interpret the results. It reminds me a bit of the networking discussions right after Mindcraft hit.

2. A Defense Of utime()

19 Aug - 25 Aug (4 posts): Why does utime() exist?

Julien Oster asked why the utime() system call existed. It didn't seem to have a purpose, since he couldn't see a case where someone would want to change the access time of a file to an arbitrary date. Mike A. Harris [*] pointed him to 'man utime', which listed it as required for conformance to SVr4, SVID, and POSIX. Theodore Y. Ts'o [*] explained, **"you want to be able to restore the access and modification times when (a) unpacking a tar file, (b) using rsync -t, or (c) mirroring an FTP directory. This is in turn necessary because various programs like make depend on the modification time being**

correct."

3. Dealing With Binary Files In The Kernel Source

20 Aug - 23 Aug (18 posts): Binary files in the kernel sources?

A straight-up standards discussion

Mike A. Harris [*] asked if there were any binary files distributed with the kernel sources. Mo McKinlay [*] found 'Documentation/logo.gif', and André Dahlqvist suggested converting it to a .png because of possible licensing issues with .gifs. But Mike corrected:

> **As long as the software used to create the GIF was licenced to create GIF's, owning and distributing actual GIF images is not in violation of the law. If it were, every web site in existance would be sued. Distributing web pages is not much different than distributing tarred up kernels.**
>
> **The real reason to convert it to a .png, is simply to boycott gif images entirely, which I totally agree with. IMHO, PNG should be the standard format for open source, used in place of .gif everywhere. I believe GNU advocates png usage and boycotts gif..**
>
> **My guess is the only reason for logo.gif is that nobody has bothered to notice it or care. Anyone care to convert it and submit a png? Sending a diff of a png wouldn't be a good idea I don't think. ;O)**
>
> **I can do the conversion if someone else hasn't allready..**

Matthias Andree [*] pointed out that GNU actually used .jpgs on its sites for browser compatibility. For translating 'logo.gif' he suggested 'web2png' and 'gif2png'. Michael Rothwell [*] asked if the .gif format had been used in this case because .gifs were simpler than .pngs, but Kai Henningsen [*] explained:

> **GIF and PNG have approximately the same functionality, with mostly these differences IIRC:**

- **GIF has thumbnail and animation support**
- **PNG is much better thought out and has a far more regular structure**
- **PNG has *far* better documentation (RFC 2083, for starters).**

If anything, I'd say PNG is simpler.

Elsewhere, Mitchell Blank Jr [*] suggested distributing the image iseally as 'logo.png.uu', but Matthias Andree said 'uuencode' was ancient history. He suggested 'base64encode' as the proper tool, though he didn't really see a need to encode it at all since the file was distributed in .tar.bz2 files. Albert D. Cahalan [*] pointed out that mime tools were either absent or buggy on most systems, and wreaked havoc with programs like 'mutt'. He concluded, **"The only alternative to uuencode would be a png-logo.c or png-logo.pl that spits out a png file without external help."**

Matthias reiterated that he saw no point in encoding the file at all, and I Lee Hetherington [*] and Brian Gerst [*] pointed out that 'diff' and 'patch' couldn't handle binaries. Matthias acknowledged this, and pitched:

Still, for uuencoding, there are far too many implementations of uuencode around (with/out checksum, with/out boundary, using blanks (0x20) or backticks (0x60)), and finally, uuencode is trashed when quoted-printable encoded.

These considerations are overcome by MIME which is well-defined. Wrong implementations of a well-defined standard may not be excused, flaky implementations of a not-so-well "standard" are to be avoided.

Note that particularly, 8 years after RFC 1521/1522 (which have not really changed in 2045..2049), there is absolutely no excuse for a lack of MIME support.

Albert replied, **"You have that backwards. The uuencode command, including the algorithm, is defined by The Single UNIX Specification, Version 2. MIME is not a UNIX**

requirement; the standard does not contain even a single mention of "MIME" or "mime". If your uuencode is broken, submit a bug report just as you would for a broken MIME encoder." Matthias asked where this spec was available, but there was no reply. (There's an online version (http://www.opennc.org/onlinepubs/7908799/) complete with search engine)

Alan Shutko [*] suggested bypassing the whole 'uuencode'/'mime' issue by simply converting 'logo.gif.' to .xpm, which was a purely text-based format. He added, **"It wouldn't be the first xpm in the source tree (xterm-linux.xpm is in there) and diffs would actually do something meaningful with it, rather than just substituting the uuencoded version of one compressed image file with another. It would even be readable without unencoding it."** Kai gave the command line 'gif2pnm < logo.gif | ppmtoxpm > logo.xpm' and added that the increased size used by the .xpm format seemed negligible in this case.

There was no reply.

4. Driver Organization; Serial Devices And X; Sharing Code; Philosophy Of Development

21 Aug - 25 Aug (108 posts): Re: [PATCH] Re: Move of input drivers, some word needed from you

This thread made some waves off-list because of Eric S. Raymond [*]'s paternal (or **grand**-paternal) criticisms of Linus. But the point of the thread should not be lost. It's about the organization of the source tree, and how that should be changed (or not changed) to keep information handy and useful, and to keep code maintainable.

In the course of discussion, Christoph Hellwig [*] submitted a patch to reorganize the driver directories, and Linus Torvalds [*] replied:

> **I have yet to understand WHY all of this is done.**
>
> **Why?**
>
> **I personally think this is a major step backwards.**

> **Moving things to be in the same directory just because it made some configuration easier. Nobody has explained to me why you couldn't just add one new configuration option, and make the affected drivers (in their regular places) dependent on that configuration option.**
>
> **Are all the SCSI drivers going to be under drivers/scsi/? No. The "normal" ones that don't have any better place for them are, but nobody has really suggested moving drivers/usb/storage around to another place just because it uses the SCSI layer.**
>
> **This patch is not going in until somebody can explain to me (in words of one syllable) why it makes any sense at all. As far as I can tell, it would be a ton more cleaner to do just**
>
> **mv drivers/usb/input.c drivers/char/**
>
> **and add a config option for it that the real drivers (wherever they may be - be they USB, firewire, joysticks, what-not) can know to disable themselves or not.**
>
> **I object to moving files around in ways that makes the tree _less_ clear. I want the kernel tree to be a nice hierarchy of drivers, filesystems, etc. I don't see the point in lumping everything together in one subdirectory just because they happen to be "input" devices, regardless of what kind of device they really are.**

Arjan van de Ven [*] asked why Linus had rejected a patch he'd sent that did exactly that. Linus replied, **"Probably because I get too much email, and because I hadn't realized how much email the whole input thing would end up generating.."** and asked Arjan to resend the patch (and Franz Sirl asked to be Cced on it as well).

Philipp Rumpf [*] also replied to Linus:

> **I have to admit I don't quite understand how**

> **drivers/ is organized right now. There seem to be at least four criteria for subdirectories of drivers/:**
>
> **drivers/<function> (sound, net)**
> **drivers/<interface> (char, block) (this isn't the same as function IMHO)**
> **drivers/<bus-the-device-is-on> (usb, sbus)**
> **drivers/<architecture> (s390, sgi)**
>
> **then there are some hybrids (usb seems to be both "devices that are on USB" and "USB controllers") and strange things like drivers/char/sysrq.c (which isn't a character device at all, and isn't really a driver either).**
>
> **I don't see any particular preference, and my impression is that new directories get added pretty much at random.**
>
> **I think I'm not the only one to be a bit confused about this.**

Linus replied that there were a few overlapping organizations which did make it confusion, and explained historically:

> **Basically, the thing originally was just "drivers/char" and "drivers/block". Nothing more.**
>
> **It became quite unmanageable fairly quickly, because by the time you have a few hundred files in a directory, "ls" no longer gives the directory listing, it just gives tons of lines scolling by too fast to make much sense of it. And configuration etc is painful.**
>
> **Right now, drivers/net is pretty horrible, for example, and we've started separating things out a bit (ie drivers/net/tulip, drivers/net/pcmcia etc to get a better hierarchy). Same thing happened to drivers/block - although in that case the splitup happened to drivers/ide, drivers/cdrom.**
>
> **drivers/char is messy, and _should_ probably contain only "fundamental char drivers" (ie things like basic tty layer stuff, ptys, /dev/zero, etc). It**

> **has too many "misc" drivers in it. Stuff like bttv, zr36120 etc. And the low-level serial stuff should probably go into drivers/serial or something.**

To Philipp's four categories, Linus felt the first three were fine, though he said the last probably should be 'arch/<xxx>/drivers'. He added:

> **I'm eager to add new directories if they help make for finer-granularity hierarchy.pThe reason I dislike drivers/input is that it seems to collapse many drivers under one, and split up something that conceptually is clearly one tree (usb). So it fails both the "split things up for clarity" and the "organize things after something simple" idea.**
>
> **Which is not to say that the drivers/ hierarchy is necessarily all that wonderful as-is.**

A number of folks replied with suggestions for ways to organize the hierarchy (for instance, Alan Cox [*] wanted 'drivers/video4linux/radio' and 'drivers/video4linux/video' which would scoop a lot of small drivers out of 'drivers/char'; but Linus didn't like 'video4linux' except as a group name, and suggested plain 'drivers/video' etc). In the course of these discussions, Linus revealed some of his thinking about mice and X; he said:

> **because we cannot reasonably do a good job for serial mice, we'll always be in the situation that the X server needs to be able to handle different mice. Which in my opinion makes it redundant in the kernel: while I know that things like GPM etc use mouse devices directly, I'm personally convinced that in the bigger picture the only thing that really matters is X.**
>
> **And X handles multiple input devices quite nicely, these days. That didn't use to be the case. So the advantage of having an input layer in the kernel is greatly diminished - and basically nonexistent if X still has to be able to do different mice.**

Alan pointed out, **"For those of us working on embedded platforms X is not the answer. In fact X is a cause of much**

mirth merriment and laughter when you try and mix even the tinyX works memory foot print with such devices." Linus replied, **"I agree - but the embedded market is completely different anyway, and isn't interested in nice generalizations. In embedded products you know exactly what your hardware is, and you code for it."**

At one point in the discussion came the exchange that caused a brief stir through the community. In the course of discussion Linus remarked, **"I would not be unhappy to have separate drivers for different chipsets,"** but Rogier Wolff [*] objected, **"It really would help reduce bugs if there were more shared code. If you split drivers, the number of users will drop. So less bugs get found."** Eric came in at this point, saying only, **"Linus, I think this is an excellent point and one you should consider."** Linus replied:

> **It's a stupid lie, and one we should ignore at all times it comes up.**
>
> **It is NOT true that "shared code means less bugs". People, wake up!**
>
> **It's true that "simple and tested code means less bugs". And quite often, code sharing will imply that the code gets (a) more complex and (b) has cases that people who make changes will never be able to test.**
>
> **No, I'm not saying that sharing is bad. Sharing is good. But anybody who makes a blanket statement like "sharing reduces bugs" is full of crap.**
>
> **Good and clear interfaces reduce bugs. Avoiding complexity reduces bugs. Using your brain before you code something up reduces bugs.**
>
> **But making a very complex driver that handles ten different variations on the same hardware? No. That does _not_ reduce bugs. It's quite often more efficient to have two people work on two drivers that are independent of each other, than to have two persons working on the same driver.**

> **Alan put it well at some point: the way to create working open source software is to minimize the amount of communication needed.**
>
> **Make good interfaces. THAT will reduce bugs.**

Jes Sorensen [*] agreed that sharing code at any price was not the thing to do, but pointed out that drivers copying excessive amounts of code from other drivers was bad, **"bug fixes to the original driver rarely propagate to the new one."** Regarding code-copying, Linus put forward:

> **Note that this too can be beneficial.**
>
> **Not for technical reasons, but for purely psychological ones. Which can often be as important as the technical ones in the end.**
>
> **The reason? People feel uncomfortable modifying other peoples drivers. This is often true even if the original author hasn't even been a very active maintainer for the last few years. So you find somebody new come in, who cares about the new cards, but is not willing to maintain the old stuff.**
>
> **So he creates a "new" driver that works for him, and in the end he may end up maintaining it a lot better than the non-existent maintenance that the old driver gets. The old driver ends up working with the old cards, and the new one gets the new ones. And in some cases this eventually gets the old maintainer involved again and things work out.**
>
> **Yes, this has happened.**
>
> **And yes, it can go the other way too. It can become a source of painful and unnecessary duplication.**
>
> **On the whole, people tend to _want_ to share, because it ends up being the easier "quick hack" in many cases. So I'm not worried about that part overmuch. I'm worried about people who share even when it doesn't make sense. And I'm worried**

> **about people having bad interfaces, which makes even sensible sharing end up as a experiment in horror.**
>
> **That's why I'm so un-interested in the "let's share" argument. I don't think that is where the problems are.**

There was no reply to this, but Eric also replied to Linus' reply to Rogier. He said Linus had missed Rogier's main point, which was not that shared code would directly lead to less bugs, but that shared code would ensure better testing of that code. He said:

> **Sharing code is a way to increase the test population for the shared parts, which increases expected test coverage on the shared parts, which is good. It's especially good because the shared parts tend to have subtler bug signatures than the unshared parts (data structure corruption is harder to diagnose than a device lockup), so the better test coverage happens exactly where it's most needed.**
>
> **Clean APIs also reduce bugs, of course, but that effect is orthogonal to the one Rogier was trying to describe. IMHO, we should strive for both.**

Wolfgang Walter and Ingo Molnar [*] both felt there were flaws with this idea. As Ingo put it, **"for a static requirements space what you say is 100% correct. Sharing must be maximized to get maximum test coverage - end of story. But in an OS that tries to be practical in an ever changing world governed by the laws of physics, it's more important to reduce algorithmical complexity, to speed up debugging and other time-constained activity that does not happen in an 'ideal world'. Part of the requirements are that we should both support new hardware, new hardware features, and should stay stable on old hardware, and we should be the fastest on the planet."** He summed up his argument with a metaphor, **"water over time flows to the lowest spot and becomes a lake - still we have 'seemingly impossible' things like rivers and waterfalls, which do not make sense in a static-only environment."**

Linus put this in his own words as well, in reply to Eric:

> **Testing only gets you so far.**
>
> **And having a driver that handles everything under the sun can easily get too complex - to the point where testing only shows that something is wrong, and actually _fixing_ the bugs is getting to be the problem.**
>
> **See Windows.**
>
> **Or see the IDE driver. It's actually on it's second generation, and people have talked about a third one. Supporting every IDE controller, whether it is PCI or not, whether it can do DMA or not, is just fairly painful. To the point where trying to fix one case tends to break another.**
>
> **No, the serial driver is not there yet. Probably never will be. There just isn't enough incentive for hardware people to do new things.**
>
> **But the "common code helps" thing is WRONG. Face it. It can hurt. A lot. And people shouldn't think it is the God of CS.**

To this last remark, Eric replied:

> **I think you're mistaken about this. Or, to be more precise, I think your remarkable native talent as an implementor has given you some unhelpful design biases that are only going to be cured by more experience.**
>
> **I'm fairly sure that experience will prove survivable for the Linux kernel and the rest of us -- but when you say things like the above, I start worrying a bit...**

Linus replied:

> **I'll give you a rule of thumb, and I can back it up with historical fact. You can back up yours with**

nada.

Simple rule of thumb:

- **sharing is good when it is true 100% sharing, no special cases, and there is nothing that has reason to avoid the sharing.**
  Example: the VM layer. TCP/IP. The VFS layer.
- **sharing is bad when it's done to 90%, and people work at trying to avoid using the functionality that some other people want.**
  Example: the SCSI layer. Parts of the IDE driver. Glibc vs kernel header files.

Quite frankly, the mentality that "we must share all common problems, whether it makes sense or not" has resulted in untold woes. Usually it starts out small. You add one more device. It obviously makes sense to just tweak the code a bit. You add one more. You'll add some special case code that only really matters for that one, and you hope that you got the other cases right.

Eventually, you'll have code that spans 5 generations of hardware, where the first generation and the last one basically share _no_ commonality except for the common heritage. And they share a lot of the code, because they have been written to do things the same way, even if it really wouldn't make much sense any more.

End result: bad organization for the new hardware, inability to sanely take advantage of the full features of the new hardware. Forcing people to leave old code in place, because it is work-arounds for bugs in hardware three generations gone. People end up having to thread very lightly because the developer probably has access to all the new stuff, but doesn't even have a way to test the old stuff any more - except by having users holler when he broke it by mistake when he added code to handle new cases.

Face it. Every once in a while, you have to start

> **afresh. Tell people that "Ok, we can't share this code any more, it's getting to be a major disaster".**
>
> **We've done this quite successfully several times. Each time it resulted in a better product. Which is exactly the reverse of what you advocate.**
>
> **Example 1: "hd.c" was left alone, "ide.c" was created. Some day, we'll hopefully leave "ide.c" alone, and create a "ata100.c" that only handles modern hardware.**
>
> **Example 2: 53c8xx driver. Which broke from the 53c7xx driver and just said "Screw it. That driver is too limited by the hardware it supports. I don't want to deal with it". See also the u14-34f driver vs ultrastor.c.**
>
> **Example 3: ne2k driver. Nope, the ne.c file was not just expanded to handle yet another card type. Yes, it shares the basic 8390 code anyway.**
>
> **This goes on and on and on. According to your logic, none of the above improvements should ever have happened. You're wrong.**

Rogier agreed, although he felt it was a question of degree. For instance, he argued, **"the fact that every serial driver in Linux has to do its own hangup and carrier detect logic stinks."** Linus agreed, and added, **"I definitely wouldn't argue about some tty layer and serial code changes."**

Meanwhile, Eric replied to Linus reply to him. To the statement that Eric had 'nada' to back up his argument, he replied, **"Yes, if twenty-seven years of engineering experience with complex software in over fourteen languages and across a dozen operating systems at every level from kernel out to applications is nada :-). Now you listen to grandpa for a few minutes. He may be an old fart, but he was programming when you were in diapers and he's learned a few tricks..."**

He went on to give the now-famous assessment of Linus:

> **I'm not arguing that splitting a driver is always**

wrong -- I can easily imagine making that call myself in your shoes, and for the exact reasons you give. I'm arguing that the perspective from which you approach this issue causes you to underweight the benefits of sharing code, and to not look for ways to do it as carefully and systematically as you ought.

When you were in college, did you ever meet bright kids who graduated top of their class in high-school and then floundered freshman year in college because they had never learned how to study? It's a common trap. A friend of mine calls it "the curse of the gifted" -- a tendency to lean on your native ability too much, because you've always been rewarded for doing that and self-discipline would take actual work.

You are a brilliant implementor, more able than me and possibly (I say this after consideration, and in all seriousness) the best one in the Unix tradition since Ken Thompson himself. As a consequence, you suffer the curse of the gifted programmer -- you lean on your ability so much that you've never learned to value certain kinds of coding self-discipline and design craftsmanship that lesser mortals *must* develop in order to handle the kind of problem complexity you eat for breakfast.

Your tendency to undervalue modularization and code-sharing is one symptom. Another is your refusal to use systematic version-control or release-engineering practices. To you, these things seem mostly like overhead and a way of needlessly complicating your life. And so far, your strategy has worked; your natural if relatively undisciplined ability has proved more than equal to the problems you have set it. That success predisposes you to relatively sloppy tactics like splitting drivers before you ought to and using your inbox as a patch queue.

But you make some of your more senior

> **colleagues nervous. See, we've seen the curse of the gifted before. Some of us were those kids in college. We learned the hard way that the bill always comes due -- the scale of the problems always increases to a point where your native talent alone doesn't cut it any more. The smarter you are, the longer it takes to hit that crunch point -- and the harder the adjustment when you finally do. And we can see that *you*, poor damn genius that you are, are cruising for a serious bruising.**
>
> **As Linux grows, there will come a time when your raw talent is not enough. What happens then will depend on how much discipline about coding and release practices and fastidiousness about clean design you developed *before* you needed it, back when your talent was sufficient to let you get away without. The code-sharing issue -- more specifically, your tendency to abandon modularization and re-use before you probably ought to -- is part of this. Andy Tanenbaum's charge against you was not entirely without justice.**
>
> **The larger problem is a chronic topic of face-to-face conversation whenever two or more senior lkml people get together and you aren't around. You're our chosen benevolent dictator and maybe the second coming of Ken, and we respect you and like you, but that doesn't mean we're willing to close our eyes. And when you react to cogent and well-founded arguments like Rogier Wolff's as you have -- well, it makes us more nervous.**
>
> **I used to worry about what would happen if Linus got hit by a truck. With all respect, I still worry about what will happen if the complexity of the kernel exceeds the scope of your astonishing native talent before you grow up.**

Linus didn't reply.

For folks interested in Andrew Tanenbaum's argument with Linus

back in 1992 (in which Rogier Wolff and Ken Thompson also participated -- to say nothing of Theodore Y. Ts'o [*] and David S. Miller [*]), it's archived in several places on the net, including the Skåne Sjælland Linux User Group (http://www.sslug.dk/artikler/Linux_is_obsolete.txt). Ironically, the decade Andrew referred to in the first post of that thread is nearing completion, and microkernels are still some distance from being truly usable (although the Hurd has advanced to the point of running X).

(Ed: *Personally, I think it's important to air these criticisms from Eric and anyone else, because they add to the overall pool of understanding. The fact that so much was made of Eric's post seems to miss the point that such communication should not be thought of as odd or abnormal. The free software and Open Source world is no stranger to differing opinions. Sometimes they lead to bitterness, but they always lead to greater understanding. I expect this one will as well.*)

5. Possible GPL Violation On Embedded Port

22 Aug - 25 Aug (6 posts): Boot strapping and ATE

Ian S. Nelson [*] had been building an embedded device out of Linux, and in the course of getting some questions answered, he mentioned, **"Hopefully by the end of this week I'll be able to bring it up all the way. At that point we should have a totally free (beer, NSC owns part of my code but they give it away for $0 when you sign an NDA) way to start Linux on a National MediaGX part which is pretty popular for web pads and embedded internet devices."** Igmar Palsenberg [*] asked, **"Isn't that a violation of the GPL ?? I assume you code is a derived work, so code must be freely available. I dont't call a NDA free."** Alan Cox [*] answered, **"If it is linked with GPL code then yes. In which case you wont be able to use it at all."** Ian had no more to say.

Discussions of other (possible or actual) GPL violations are covered in Issue #8, Section #13 (**27 Feb 1999:** "Possible GPL Violation By Mosix"), Issue #74, Section #14 (**23 Jun:** "Anonymous Poster Claims GPL Violations Are Accepted By Linux Community"), and Issue #79, Section #7 (**20 Jul:** "'Virgin Connect' Violates GPL").

6. 2.4.0test7-pre7

22 Aug (1 posts): 2.4.0test7-pre7

Linus Torvalds [*] announced:

> **Ok, getting close to the real test7.**
>
> **The patch looks bigger than it is due to the drivers re-organization (discussed to death on this very list ;).**
>
> **pre7:**
>
> - **block_all_signals()/unblock_all_signals() interface to allow drivers to react nicely to signals that happen to a process that owns driver data. Read: direct rendering lock handling.**
> - **ThunderLAN update (timer fixes, full-duplex, activity-led)**
> - **Fix NFS oops on removing negative dentry. Honour rsize for directory read.**
> - **usb updates**
> - **scheduler wakeup race fix.**
> - **move radio/tv cards to drivers/media, cleaning up drivers/char**
> - **move "input" layer to drivers/input, cleaning up drivers/usb**
> - **Cirrus SoundFusion CS4280/461x sound driver.**
> - **proper camera locking in usb/dc2xx.c**
> - **USB printer driver update (Printer Protocol 3 and timeout handling)**

There was no reply.

7. BeFS For 2.2 And 2.4

22 Aug (7 posts): PATCH: BeOS FS support for 2.2.16

Interesting discussion (with a twist at the end), the upshot being that BeFS is coming to the kernel. Miles Lott [*] announced:

> **I took some time to update the code I found at http://hp.vector.co.jp/authors/VA008030/bfs/ (http://hp.vector.co.jp/authors/VA008030/bfs/). This patche applies to a clean 2.2.16 kernel tree and adds support for bos (name changed to prevent conflict with bfs in 2.4). I hope to mod it for use in 2.4 as time permits.**
>
> **It provides for read-only, however it will allow you to mount read write. Not certain that it won't do something nasty, so mount with -o ro.**
>
> **Basically, I fixed one typo and updated the nls stuff for long filename support.**

David Weinehall [*] was happy to hear about this, and liked the patch itself. He invited Miles to port to 2.4 as well, but he suggested "befs" as a more natural name than "bos". Miles replied he was working on the 2.4 port, but that most filesystems seemed to have 3-letter names; but David listed:

> **Ehrmm...**
>
> **Let's see (this is from the v2.4-test tree, not v2.2.xx):**
>
> - **ntfs**
> - **hpfs**
> - **ext2(fs)**
> - **qnx4(fs)**
> - **ramfs**
> - **romfs**
> - **umsdos**
> - **vfat**
> - **sysv(fs)**
> - **coda**
> - **adfs**
> - **affs**
> - **cramfs**
> - **isofs**
> - **jffs**
> - **ncpfs**

> **Not to be picky, but I'd say there are more 4-letter abbrevations than three letter abbr's.**
>
> **befs (not beos; the fs should be there to stress that it is a filesystem) is a good name I think. beosfs might be a good name too.**

(He added that he was happy Miles was working on a 2.4 port). Miles agreed that 'befs' would be fine (he said he'd been locked into a 'bfs', 'bos', 'fat' trichotomy), but Bjorn Wesen [*] came in at that moment to say, **"FWIW, the book 'practical file system design' says its bfs, and that was written by a kernel engineer at Be (not saying Linux cant use any name we want, but at least they seem to call it bfs...)"** There was no reply.

8. Linus On Flags In Functions To Enable Hardware Features

24 Aug - 25 Aug (7 posts): PATCH - final (I hope) tidy up for generic_make_request interface

Neil Brown [*] posted a patch to change several things, but Linus pointed out that certain hardware had the ability to do part of Neil's functionality on its own, and Linus explained, **"that part of your change would make such a driver unable to take advantage of the resources offered by the hardware."** Neil had been acting based on a comment in the current code that had seemed to indicate that the existing code was just a kludge, to be replaced in 2.5 by something better; but he said he'd undo that part of his change and resubmit the patch. Linus pointed out that they'd have to have some clean way to handle old drivers that didn't know about the new functionality, and Neil suggested, **"Maybe we should then have a "blk_queue_highmem_capable(q,flag)" function whereby drivers can tell elevator_make_request that they are highmem aware, and so the create_bounce will be skipped ??"** Linus replied:

> **No. No flags. That's the mistake SCSI did, and it is a fundamental mistake. It makes it basically impossible to "shift out" old code.**

> **I'd much rather have two separate functions for this - sure, they can share most of the code by using common inline functions and what not, but I'd be a lot happier with drivers just using the function that they are most comfortable with directly.**
>
> **Anyway, this is not a change I'd like to have at this point. This is definitely a 2.5.x thing.**

Neil posted a new patch, but there was no further discussion.

9. Nearing 2.2.17

24 Aug - 26 Aug (6 posts): Linux 2.2.17pre20 - the for Linus edition

Alan Cox [*] announced:

> **This is the version waiting Linus. Its in the queue for holy penguin pee so either it will get peed on or I will get abuse from Linus depending whether he likes it or not 8)**
>
> **In the mean time I'm taking a break rescuing a rather buggy but otherwise neat user space app. I'll start 2.2.18 in a couple of weeks.**
>
> **James Cloos is making diffs between 2.2.17pre versions are available at http://jhcloos.com/pub/linux-2.2.17-pre (http://jhcloos.com/pub/linux-2.2.17-pre) ftp://jhcloos.com/pub/linux-2.2.17-pre (ftp://jhcloos.com/pub/linux-2.2.17-pre)**
>
> **2.2.17pre20**
>
> 1. **Fix EIP/ESP printk thinko (Willy Tarreau)**
> 2. **Final small DAC960 adjustments for 2.2.17 (Leonard Zubkoff)**
> 3. **Improve AARP handling (Alistair Riddell)**
> 4. **Fix bug in the appletalk code (Marcelo Tosatti)**
> 5. **Last minute fix to the CS4281 (Tom Woller)**

6. **Mention CS4280 and use __initdata for data (me)**
7. **Turn on the EAPD bit on the AD1885 (me)**
8. **Don't honour the status bit for audio on a 440MX - it appears it doesnt work (Marcus Sundberg)**
9. **Fix high cpu usage on i810 audio (Marcus Sundberg)**
10. **Apply the same fix to the cs46xx (Bill Nottingham)**
11. **Change the power/CD algorithm on the 46xx (me)**

Tom Leete [*] reported, **"This is the first 2.2.x I've built since pre13. Am I the only one who got a surprise when depmod -a took 20 minutes to run? There was a big spill of 'not an ELF archive' messages, but I got a nice dependency tree for all the kernel object files. Too bad it confuses modprobe so. I suppose there's a modules.conf trick to exclude [build]."** Sven Koch and Horst von Brand [*] suggested upgrading modutils.

10. New Maintainer For VIA vt82c586 IDE Driver

25 Aug - 27 Aug (8 posts): UDMA trouble in VT82C586 IDE

Cesar Eduardo Barros [*] was having some trouble getting UDMA working with his FIC SD11 motherboard. Enabling via82cxxx tuning or enabling UDMA via 'hdparm', he'd get timeouts waiting for DMA during boot. Apparently Vojtech Pavlik [*] sent him some old patches in private, because he replied to himself the next day saying the problem was solved. He suggested including the patches in future kernels, but Andre Hedrick [*] replied, **"All Vojtech has to do is accept responsiblity for the chipset code. I mean all flavors of the bastardization of the design. This includes the ones that you can change the chipset reporting type if you know the dirty-secrets. The problem is that some mainboard manufacturers alter the PCI ID codes to fake compatablity."** He added, **"In the past Vojtech was not ready to jump on this and run. I have one, and only one, completely corrupted drive, beyond recovery by any means, because of one incomplete thought in the design. I**

will not accept this happenning again." Vojtech replied:

> **Now, after the last MWDMA timing issue was resolved, I can do that. Yes, while I can't say my code is 100% perfect, because I'd have to verify that with an IDE analyzer which unfortunately I don't have access to, after some time of extensive testing there are no more failure cases on all the machines it was tested on. (One case is still unclear, but that user is now using 2.2 and couldn't do more testing.)**
>
> **If you need it said explicitely - yes, now I can stand behind that code and I do accept the responsibility for it if it breaks anywhere. (Of course the legal no-warranty statement still applies).**

He posted a patch against 2.4.0-test7, asking Linus Torvalds [*] to accept it if Andre approved it. Andre replied, **"Since you did not hesitate this time as the first time I asked the question, DO IT! Vojtech, this time I trust that you did it right! Good Job!"** He approved the patch as a replacement for the current code, and asked Linus to apply it.

Mark Rutherford asked if this would be backported to 2.2, but Andre said no.

11. Compiling Kernels With 'kgcc'

27 Aug - 28 Aug (7 posts): Request for information.

Henri J. Schlereth asked what he had to do to compile a 2.2.16 kernel with 'kgcc' instead of 'gcc'. Did he just have to change the "HOSTCC=gcc" line in the Makefile, or were other changes required? Igmar Palsenberg [*] cautioned that 'gcc' was the only tested compiler, and Michael Elizabeth Chastain [*] also explained:

> **Building the kernel uses two compilers. CC is the compiler for actually building the kernel you are going to boot, and HOSTCC is the compiler for building auxiliary programs *while you are building the kernel*.**

> **So you don't need to change HOSTCC.**
>
> **The host side programs compiled with HOSTCC include the configuration programs and other programs in scripts/, as well as some table generation programs in places like drivers/char and drivers/sound.**

David S. Miller [*] suggested the command line 'make CC=kgcc bzImage', but Peter Samuelson [*] and Horst von Brand [*] pointed out that this would only work for development kernels as of 2.3.35 or so, not for 2.2.16. Alan Shutko [*] suggested that Henri ask on the Redhat pinstripe list, since as far as he knew that was the only distribution using 'kgcc'.

12. Time To Replace The Big Kernel Spinlock With A Semaphore?

28 Aug (5 posts): [patch] getting rid of the Big Kernel Spinlock, 2.4.0-test7

Ingo Molnar [*] proposed:

> **during 2.3 we got rid of 99% of lock_kernel()s within the core kernel. IMHO the time has arrived to get rid of the big kernel spinlock forever - by changing it to a ordinary semaphore. Most lock_kernel() code paths are holding the kernel lock for a long time and they do it rarely - so a semaphore is more suited to do this than a spinlock. It's even a speedup (and a debugging help), because lock_kernel() will not spin wasting CPU cycles anymore.**
>
> **the attached biglock-2.4.0-test7-A6 patch implements this and does a couple of related cleanups:**
>
> - **the impact of the 'big IRQ lock' is smaller as well, thus schedule() now enforces that it is to be called with IRQs turned on. sleep_on() enables IRQs, and entry.S's reschedule as well. (the later is a code path taken by**

> **returning IRQ handlers which might not always disable IRQs.)**
>
> **As a result neither release_kernel_lock(), nor the tq_scheduler path needs to do a sti() - which speeds up the common schedule() path. Nor the performance, nor the semantics of sleep_on() are impacted.**
>
> - **new sema_locked() added to semaphore.h - similar to is_spin_locked().**
> - **got rid of asm/smp_lock.h - linux/smplock.h has the generic code, now that no global-IRQ locking logic is present in schedule(). This simplifies things.**
> - **irqs_enabled() added to asm/hardirq.h.**
>
> **the only place that still needs to spin actively is reacquire_kernel_lock() - because semaphore.c calls schedule() itself, so we cannot call down() at that point. Solving this is not impossible, but needs more changes than justified i think.**
>
> **the patch has the IMHO nice side-effect of 'speeding up races' - more processes can run when the kernel lock is taken because there is no active spinning - thus SMP, kernel-lock related races are triggered with higher probability.**
>
> **the patch compiles/boots/works just fine on SMP and UP x86 systems. Other architectures have to add the (trivial and generic) sema_locked() and irqs_enabled() functions. The patch should be an invariant and does not intend to change semantics of any kernel functionality otherwise.**

Alexander Viro [*] objected to the whole idea of changing the big lock into a semaphore, explaining, **"right now BKL can be acquired while we are holding a spinlock. It's a bad idea, but it's possible. With your change it becomes deadly. Another reason: currently BKL can be taken in the middle of operations on per-CPU data. It will not block, so nothing will get that CPU while we are in lock_kernel(). Not true with your patch."**

Ingo Molnar hadn't noticed these problems, and was very happy to hear about them, but asked if it wasn't a bug to acquire the BKL while holding a spinlock. It seemed extremely dangerous to him. Alexander agreed it should be fixed. Ingo also thought the other objections might be bugs as well, and they went back and forth for a bit on it.

It looks like the world may not be ready to lose the big lock just yet...

We Hope You Enjoy Kernel Traffic