Kernel Traffic #87 For 2 Oct

By Zack Brown

Table Of Contents

Introduction

Thanks go to Chris Beckenbach, Michael Cope, and Jeff Gage For catching an unexpanded <kcref> (inter-issue reference) last week in Issue #86, Section #16 (**16 Sep:** "NFS Patches In 2.2; Yes, 2.2"). Thanks, guys! Jeff additionally caught part of a quote in Issue #86, Section #9 (**12 Sep:** "Using Netware Elevator Ideas In Linux") that was surrounded by a < and >, rendering that text invisible. Good catch! Those are really tough bugs to spot. Thanks!

Thanks also go to Peter Samuelson for catching me inadvertantly sticking extra text into a quote by Alan Cox [*] in Issue #86, Section #8 (**12 Sep:** "Backporting The IDE Patch"). I've fixed it so that at the bottom of that section, Alan no longer seems to direct people to a different KT section ;-). Thanks, Peter!

Mailing List Stats For This Week

We looked at 1594 posts in 5945K.

There were 402 different contributors. 203 (50%) posted more than once. 161 (40%) posted last week too.

The top posters of the week were:

- 73 posts in 283K by "Jeff V. Merkey" <jmerkey@timpanogas.com>
- 61 posts in 234K by Linus Torvalds <torvalds@transmeta.com>
- 58 posts in 159K by Alan Cox <alan@lxorguk.ukuu.org.uk>
- 42 posts in 145K by Alexander Viro <viro@math.psu.edu>
- 41 posts in 160K by "Andi Kleen" <ak@suse.de>

1. Possible GPL Violations By Microsoft; Kernel Debugger In Official Sources

2 Sep - 19 Sep (376 posts): [ANNOUNCE] Withdrawl of Open Source NDS Project/NTFS/M2FS for Linux

In the course of discussion, Jeff V. Merkey [*] explained why he couldn't release certain code. He said, **"If we post it, Microsoft will grab it and it will be in NT within 48 hours of them downloading it from our site."** Andre Hedrick [*] replied, **"I know what you mean about microsoft. My and co-worker's code for doing full taskfile access under linux was rejected here but is being used in MicroSoft Whistler 2001. They are quick to grab the very best of Linux and adopt it for their own."** Henning P. Schmiedehausen [*] suggested going to the FSF about the GPL violations, but Andre replied that if patches were rejected from the kernel, it wasn't a GPL issue and MS could do what it liked. But Theodore Y. Ts'o [*] pointed out, **"That's B.S. The GPL is a Copyright license; it applies whether or not it is in the kernel. Microsoft (or anyone else for that matter) can't take your code and use it without consent. The GPL is one way of giving consent, with certain strings attached."** As Andrew McNabb [*] put it elsewhere, **"You automatically own an implicit copyright on anything you create. Regardless of whether you specify a license for code you write, anyone who steals it is breaking the law. The GNU license, if specified, allows people to use your code if they agree to your terms."**. In response to Ted, someone mentioned that no one would have the money to go after MS for those violations, but Henning said, **"The FSF will surely step up (that's what RMS dreamed all about since he started GNU. :-))."** [...] **"I'm sure that once the FSF is willing to step up, there will be lots of supporters and sponsors to finance this."**

Elsewhere, and on a completely different topic, Jeff complained about the absense of a kernel debugger. Jes Sorensen [*] said that 'kdb' had existed for quite awhile, and Jeff vented:

> **KDB is putrid. Can it debug double faults? NO. Can it debug complex register and numeric evaluation statements like IF ((EAX == 1) && [ESP-4] == 0x3000)? NO. Can it debug nested task gate exceptions? NO. Can it debug SMP locks races? NO. Can it debug priority inversion problems in sleep locks? NO.**
>
> **Can the Kernel debugger in NetWare? YES. Can the Kernel Debugger in NT? YES.**

Alan Cox [*] replied:

> **Remote gdb on Linux - yes and I can do my debugging source level. Unfortunately Linus seems to have a one man campaign against putting sensible debugging into his kernel.**
>
> **The tools exist and they should be in the x86 tree as well as sparc etc where with other maintainers it is present**

Jeff speculated, **"I can only assume the reason for this is one of control, and that there are no valid technical reasons for it. I have spent more nights with printk() than I care to."** And David S. Miller [*] replied:

> **And I bet the lessons learned and the issues involved in those nights with printk will never leave your brain, you will remember precisely in the future next time you see the same types of symptoms what kinds of things to look for and where.**
>
> **This is what a debugger does not do for you. The debugger allows you to be lazy, step around, "oh yeah check for NULL" and never have to _think_ about what you're doing or the changes you're making or even if the same bug might be elsewhere.**
>
> **This is why Linus does not allow a debugging facility like this into the kernel, so people spend time _thinking_ when they go hunting down bugs.**
>
> **It takes longer to nail a bug, yes, but the resulting fix is always far superior. And the person who discovers the bug leaves with a much larger amount of knowledge about how that area of the kernel works.**

Alan replied, **"There are only a few things I think Linus is a complete loon about 8) but the debugging stuff is one."** David disagreed with this, and added, **"I take much comfort in the fact that 2 hackers who have been debugging programms probably longer that I have been alive (Kernighan and Pike)**

agree with me. See chapter 5 of their book "The Practice of Programming". Note in particular the second paragraph on page 119."

Elsewhere, Ingo Molnar [*] gave his take on kernel debugging:

> **There are two basic types of 'debugging behaviors':**
>
> > **A) 'Collect enough data to understand the local context and fix the bug.'**
> >
> > **B) 'See the point of the immediate crash, think about that code, think about code that called this code, think about the intent and implementation of that code and find the bug based on understanding the code.'**
>
> **A) results in people being able to debug easy and moderate kernel bugs. The same people are lost if faced with bugs that are not apparent in the 'state of the system around the bug' represented by the debugger.**
>
> **B) results in people with the ability to identify bugs based on the source code - and the ability to fix the really mindblowing tough bugs. We had about 10 such bugs during the cycle of the 2.3 kernel.**
>
> **I do claim that the real mainsteam-kernel development 'bottleneck' is the ability to fix the tough bugs. It's always these tough bugs that keep up the addition of new features in devel kernels. We should thus do everything to teach people the proper debugging methodology.**
>
> **A) takes less time for the easy bugs - it might also speed up driver development. But it is utterly useless for the really tough problems. We had really tough bugs in 2.3 that one couldnt find even with the heaviest hitting debugging equipment. Debugging code in the kernel also blurs the intention of the code, which makes B) harder.**
>
> **B) is a slower process to both learn and practice, but will also lead to more (unrelated) code improvements (based on better understanding of various kernel subsystems) and ultimately leads to better kernel developers.**
>
> **one problem is developer laziness ;-) We could as well include debugging code so that experienced people like you can fix easy / moderate bugs quicker. But the problem is that in this case people are not forced to understand the code as much as if the debugging tools were 'minimalistic' like today.**

> **i have nothing against (in fact contributed to) independent debugging functionality like IKD / KDB. It's slightly more pain to keep it uptodate with the devel kernel, but the devel kernel itself must stay focused on the strategic goals, not the tactical ones.**

Andi Kleen [*] said he didn't think that omitting useful debugging tools would make anyone less likely to use ingo's "A" approach, and Ingo replied, **"well, they will sooner or later notice that it's easier to fix bugs by following the development of the kernel and understanding the underlying principles and the code. As elitist as it might seem, we rather need 10 highly skilled developers who understand the kernel than 100 moderately skilled ones who know how to operate a kernel debugger and barely anything else. [this is not an insult towards people with less experience - having less experience is just a temporary stage of a process, nothing else. But if it becomes the end station thats a problem IMHO.]"** Richard Gooch [*] replied, **"I think there's a certain amount of wishful thinking here. As you say, everyone has to start from nothing. But if the learning/development curve is too steep, or the process is too frustrating, you are going to lose a proportion of the potential gurus. You can't push people in a direction they don't want to go."** Ingo replied that anyone was free to apply the 'kdb' and other debugging patches themselves.

At one point farther down in the thread, Linus Torvalds [*] finally got involved. He stated his position:

> **I don't like debuggers. Never have, probably never will. I use gdb all the time, but I tend to use it not as a debugger, but as a disassembler on steroids that you can program.**
>
> **None of the arguments for a kernel debugger has touched me in the least. And trust me, over the years I've heard quite a lot of them. In the end, they tend to boil down to basically:**
>
> - **it would be so much easier to do development, and we'd be able to add new things faster.**
>
> **And quite frankly, I don't care. I don't think kernel development should be "easy". I do not condone single-stepping through code to find the bug. I do not think that extra visibility into the system is necessarily a good thing.**
>
> **Apparently, if you follow the arguments, not having a kernel debugger leads to various maladies:**
>
> - **you crash when something goes wrong, and you fsck and it takes forever and you get frustrated.**
> - **people have given up on Linux kernel programming because it's too hard and too time-consuming**
> - **it takes longer to create new features.**

And nobody has explained to me why these are _bad_ things.

To me, it's not a bug, it's a feature. Not only is it documented, but it's _good_, so it obviously cannot be a bug.

"Takes longer to create new features" - this one in particular is not a very strong argument for having a debugger. It's not as if lack of features or new code would be a problem for Linux, or, in fact, for the software industry as a whole. Quite the reverse. My biggest job is to say "no" to new features, not trying to find them.

Oh. And sure, when things crash and you fsck and you didn't even get a clue about what went wrong, you get frustrated. Tough. There are two kinds of reactions to that: you start being careful, or you start whining about a kernel debugger.

Quite frankly, I'd rather weed out the people who don't start being careful early rather than late. That sounds callous, and by God, it _is_ callous. But it's not the kind of "if you can't stand the heat, get out the the kitchen" kind of remark that some people take it for. No, it's something much more deeper: I'd rather not work with people who aren't careful. It's darwinism in software development.

It's a cold, callous argument that says that there are two kinds of people, and I'd rather not work with the second kind. Live with it.

I'm a bastard. I have absolutely no clue why people can ever think otherwise. Yet they do. People think I'm a nice guy, and the fact is that I'm a scheming, conniving bastard who doesn't care for any hurt feelings or lost hours of work if it just results in what I consider to be a better system.

And I'm not just saying that. I'm really not a very nice person. I can say "I don't care" with a straight face, and really mean it.

I happen to believe that not having a kernel debugger forces people to think about their problem on a different level than with a debugger. I think that without a debugger, you don't get into that mindset where you know how it behaves, and then you fix it from there. Without a debugger, you tend to think about problems another way. You want to understand things on a different _level_.

It's partly "source vs binary", but it's more than that. It's not that you have to look at the sources (of course you have to - and any good debugger will make that _easy_). It's that you have to look at the level _above_ sources. At the meaning of things. Without a debugger, you basically have to go the next step: understand what the program does. Not just that particular line.

> **And quite frankly, for most of the real problems (as opposed to the stupid bugs - of which there are many, as the latest crap with "truncate()" has shown us) a debugger doesn't much help. And the real problems are what I worry about. The rest is just details. It will get fixed eventually.**
>
> **I do realize that others disagree. And I'm not your Mom. You can use a kernel debugger if you want to, and I won't give you the cold shoulder because you have "sullied" yourself. But I'm not going to help you use one, and I wuld frankly prefer people not to use kernel debuggers that much. So I don't make it part of the standard distribution, and if the existing debuggers aren't very well known I won't shed a tear over it.**
>
> **Because I'm a bastard, and proud of it!**

There were a lot of replies to this, and a number of smaller subthreads branched off, with everyone falling strongly on one side or the other.

2. Wine In The Kernel; GPL Loopholes

7 Sep - 19 Sep (64 posts): [RFC] Wine speedup through kernel module

David Howells [*] announced:

> **I've done an implementation of some of the Win32 "system calls" in a kernel module in an attempt to speed up Wine.**
>
> **The preliminary benchmarks that I made, while not very real-world since I don't think I have managed to implement enough for that yet, seem to indicate that in some tests, I can beat Win2000 by 20% on syscall latency, and Wine by 900%.**
>
> **Wine is this slow, I think, because it uses an extra process (the wineserver) to implement synchronisation and handle<->fd mapping, and involves the use of sendmsg and recvmsg to communicate.**
>
> **I'm interested in your comments on the code I've written (attached to the bottom of this document). I'm also interested in ideas as to how a large speed up might be implemented by some other user-space mechanism or by less intrusive kernelspace intervention.**
>
> **I'm also interested in finding a better way of getting to kernel space from user space... Currently, this involves the client process opening a proc file and doing ioctl's on it to request Win32 operations (easy to do from a kernel module).**

> **However, the ioctl syscall seems to incur a fairly large penalty in the number of standard I/O commands it checks for first, before calling the supplied service routine.**
>
> **Other methods that have been mentioned include persuading Linus to reserve a syscall number specifically for this purpose, and having the module supply a handler for it.**

Martin Dalecki [*] objected harshly:

> **Please by no way don't include this patch into the official tree. It's insane due to the following:**
>
> 1. **Linux is UNIX not NT... (in terms of API)**
> 2. **WINE in itself is barely usefull - even in fact non existant, since there is no official stable release out there.**

Adam Sampson [*] agreed, though added soberly, **"I also don't think this patch should go into the official tree, but in my case it's because it's too closely related to the WINE userspace libraries; it should become part of the WINE tree instead. (Note that this is how other packages such as ALSA and lm_sensors are maintained.)"** He (and others) also objected that Wine was very useful, and shouldn't be put down just for being pre-1.0

Albert D. Cahalan [*] was not opposed to David's idea, and remarked, **"If you implement enough to run the common Windows benchmarks, we can have loads of fun."** Elsewhere, also in reply to David's initial proposal, Linus Torvalds [*] replied:

> **Hmm.. I have this feeling that it would be much nicer to just implement the NT system calls directly.**
>
> **We used to have the iBCS2 project, and I was actually considering making it part of the standard kernel when it started becoming a non-issue simply because there started to be more native Linux programs than iBCS2 programs.**
>
> **And we've already had the vm86 mode and the BIOS trap stuff speedups for DOSEMU for a long time.**
>
> **It looks like if you want to do this, it would be better to just try to do it outright, and have the same kind of "emulate the ones we know about and care about performance" in the kernel that we already have with the vm86 mode emulation.**
>
> **I wouldn't be adverse to supporting Wine better - as long as it's fairly cleanly separated. This doesn't look too bad.**

There followed a technical discussion, during which there arose the possibility of dynamically allocated syscall entries. There was a bit of back-and-forth on this, until Linus came down on it, saying:

> **The problem is that dynamic system calls are not going to happen.**
>
> **Why?**
>
> **License issues. I will not allow system calls to be added from modules. Because I do not think that adding a system call is a valid thing for a module to do. It's that easy.**
>
> **It's the old thing about "hooks". You must not sidestep the GPL by just putting a hook in place. And dynamic system calls are the ultimate hook.**

Pavel Machek [*] pointed out, **"Well, you can't stop _GPL-ed_ modules from adding dynamic system calls."**

KT previously covered the possibility of this kind of loophole in Issue #8, Section #13 (**27 Feb 1999:** "Possible GPL Violation By Mosix"), in which Richard Stallman also made some pointed remarks.

3. Uncertainty About PowerPC Port Maintainership

13 Sep - 23 Sep (44 posts): PowerPC Linux for AS/400 & RS/6000 Hardware

Dwayne Grant McConnell [*] said to Linus (and a bunch of other folks, and a couple lists), **"Cort Dougan [*] recently announced he was no longer going to be maintaining the PowerPC Linux tree. There is a team within IBM actively working on PowerPC Linux for both 32-bit and 64-bit hardware and we are very interested in continuing the development of PowerPC Linux. We have been working with Paul Mackerras [*] and others on the current set of issues for PowerPC Linux and future directions. We have been preparing to announce our PowerPC Linux trees and should do so within the next few days. Could you please add the following changes to the MAINTAINERS file indicating we are willing to maintain PowerPC Linux for AS/400 and RS/6000 hardware?"** He included a patch to add himself and Tom Gall [*] to the MAINTAINERS list, as taking responsibility for the port. But Paul Mackerras replied, **"I believe what Cort said is that he is no longer maintaining the http://www.ppc.linux.org/ (http://www.ppc.linux.org/) web site. I don't think he meant that he was no longer maintaining Linux for PowerPC. He is away at the moment though."** Dan Malek [*] had the same impression, and replied:

> **I met with Cort for a couple of hours on Tuesday. I thought he was going to be back in NM sometime today (Wednesday), and I was hoping he would have responded to the messages of confusion by now.**

We discussed how to better manage the kernel source trees, and it was pretty clear to me he intended to keep working on this. Like all of us, he has other jobs to do, and sometimes he just can't respond as quickly as we like. This kind of work has a multiplier effect as it moves upstream.

Meanwhile, Cort Dougan also replied to Paul, **"What I'd said was I'm taking a vacation from maintaining the PPC-tree for a while so I can do some of my real job. I'm not abandoning it in any way - just spending less time on it for a while."**

The discussion skewed off into other stuff at this point.

4. More Kernel Debugger Advocacy

14 Sep - 15 Sep (31 posts): The case for a standard kernel debugger

Keith Owens [*] proposed:

This note puts the case for including a kernel debugger in the master tarballs. These points do not only apply to kdb, they apply to any kernel debugger. Comments about the perceived deficiencies of kdb, kgdb, xmon or any other debugger are not relevant here, nor are questions about how or when debuggers should be activated. I want to concentrate of whether the kernel should have *any* standard debugger at all.

If Linus still says "no" to including any debugger in the master tarball then that will be the end of this thread as far as I am concerned. I will then talk to distributors about getting a debugger included in their kernels as a patch. Hopefully the distributors who want a kernel debugger can then agree on a standard one.

Disclaimer: Part of my paying job is to maintain kdb. SGI want kdb to be used more widely to benefit from GPL support. More eyes and hands means better code for everybody.

1. **Random kernel errors are easier to document and report with a debugger. Oops alone is not always enough to find a problem, sometimes you need to look at parameters and control blocks. This is particularly true for hardware problems.**
2. **Support of Linux in commercial environments will benefit from a standard kernel debugger. The last thing we want is each commercial support contract including a different debugger and supplying different bug reports. Bug reports on supported systems should go to the support contractor**

but some will filter through to the main linux lists.

3. **Architecture consistency. Sparc, mips, mips64, ppc, m68k, superh, s390 already have remote debugger support in the standard kernel. i386, alpha, sparc64, arm, ia64 do not have standard debuggers, they have to apply extra patches. Some architectures have extra debugger code in addition to the remote gdb support.**
4. **Debugger consistency. Back in 1997 there were a lot of individual kernel debugging patches going around for memory leaks, stack overflow, lockups etc. These patches conflicted with each other so they were difficult for people to use. I built the original set of Integrated Kernel Debugging (IKD) patches because I contend that having a standard debugging patch instead of lots of separate ones is far easier for everybody to use. The same is true of a kernel debugger, having one standard debugger that all kernels use will improve the productivity of everyone who has to support kernel code, no need to learn the semantics of multiple separate debuggers.**
5. **Easier for kernel beginners to learn the kernel internals. Having worked on 10+ operating systems over the years, I can testify that some form of kernel/OS tracing facility is extremely useful to get people started. I agree with Linus when he said**

   "'Use the Source, Luke, use the Source. Be one with the code.'. Think of Luke Skywalker discarding the automatic firing system when closing on the deathstar, and firing the proton torpedo (or whatever) manually. _Then_ do you have the right mindset for fixing kernel bugs."

   But Linus has also said "The main trick is having 5 years of experience with those pesky oops messages ;-)". Beginners need some way of getting that experience. Reading the source from a cold start is an horrendous learning curve, debuggers help to see what the source is really doing. Always remember that 90%+ of kernel users are beginners, anything that helps to convert somebody from kernel beginner to kernel expert cannot be bad.
6. **I contend that kernel debuggers result in better patches, most of the time. Sometimes people misuse a debugger, as Linus said**

   "I'm afraid that I've seen too many people fix bugs by looking at debugger output, and that almost inevitably leads to fixing the symptoms rather than the underlying problems."

   Does that occur? Of course it does, I have been guilty of that myself over the years. Is it inevitable? IMNSHO, no. Seven of

> > **the twelve architectures in the standard kernel already have built in debuggers. Where is the evidence that these architectures have more bad patches because of the presence of the debuggers?**
> >
> > **Even if somebody does submit a patch to fix the symptom instead of the problem, that alone is valuable information. Fixing the symptom focuses attention and the associated information helps to fix the real problem. We get problem patches even without debuggers (let's not mention the recent truncate problems ;) but there are enough eyes on the kernel to find problem patches and remove them. Adding a standard debugger will improve the quality of some of those eyes at a faster rate.**
>
> **So how about it Linus? Does any of this change your mind about including a standard kernel debugger?**

Jeff V. Merkey [*] gave a happy yell and threw his fist in the air, as did David P Howell; and Richard Moore [*] added:

> **I think the case for the kernel debugger is better stated as the case for RAS (Reliability, Serviceability and Availability) in the kernel, in other words, there is a case for having the right diagnostic, reporting and recovery tools in the right place at the right time. A kdb does not fulfil all diagnostic RAS needs. IMHO it's an extremely powerful developement tool, but hang on so is a logic analyser and a source-level debugger. It can also be a real pain if trying to debug HLL source using an assembler based debugger. The point is, one generally needs a debugger that matches the semantics that the programmer is dealing with. If its, assembler code the so be it, use a kdb. If you're poking around with H/W specific interfaces and system busses the you make need a lower level tool.**
>
> **But should a kdb be a standard part of the kernel for use in production/commercial/enterprise environments? I don't believe so. Looking back at the techniques we've deployed over the years to debug system problems in commercial environments, we only ever had the luxury of using a kdb with OS/2. Just about every other OS we supported did not have a kdb. The OS/2 case is interesting because initially, we had only the kdb for debugging, and it was the worst platform for serviceability that we ever supported. We couldn't debug those typically obscure problems that occurred only in production environments and which could never be readily re-created in the lab. We took an enormous amount of pain over this from our customers over poor serviceability. They hated every minute of production time we took from them when a developer took control of their systems in order**

to debug, or in many cases not debug the problem. Of course we had created a rod for our own backs. Customers knew we never sent developers on site to debug OS/390 (or MVS as it was called in those days). They also knew that we never rejected a problem because it was not re-creatable and we never even asked for a re-creation scenario. The reason for this was that we had appropriate RAS capability in MVS which allowed data to be captured automatically at fault time combined with a certain amount of self-healing capability and automated recovery. What we did to OS/2 to make it approach this level of RAS capability was to implement a system dump capability - similar to SGI's kernel crash dump, + a comprehensive system tracing facility that could be dynamically customised to tracing events in any code path without any code recompilation - IBM's Dynamic Probes for Linux is an initial port of the capability + a comprehensive and customisable virtual storage based dump, a bit like core dump, except that it could dump process trees if required and memory from not only from user space, but from system space based upon kernel sub-component, for example file-system structures etc..

That capability completely transformed our ability to debug serious and obscure problems, with minimal disruption. It's true that we weren't immediately successful when we implemented this stuff. There's a major learning curve and mind-set change required to work with captured data as opposed to interactive debugging.

We didn't throw away the kdb, it's still very useful:

1. **as a didactic tool.**
2. **for the final stages of problem determination - every problem is re-creatable once you know the triggers. And when you do, which you can get from dumps and traces, then you can set up a lab-based experiment where you use a debugger to solve the final mystery.**
3. **in production for those exceedingly rare cases where we needed to know what the underlying hardware was up to - it's a cheaper option than using a logic analyser.**

One big argument against RAS of any sort is that it bloats the kernel and not every one wants it (until they have a problem). A further argument with Linux is that you may have to do quite a bit of hard work to get the subset of RAS you need to co-exist, if it exists at all. Something we're working on which may help resolve this, and will be made available with the next drop of Dynamic Probes is Generalised Kernel Hooks Interface (GKHI). The idea here is to make all our RAS function the option of being dynamically loadable kernel modules. In most cases we don't

> **need to modify kernel function, just get control at the right time. So we place hooks in kernel source, which remain dormant until activated by the GKHI when a RAS module asks it to. Maybe this will provide a way out of the difficulty.**

Timur Tabi [*] also approved of Keith's proposal, but added that he could relate to Keith's 5th point the best: using a debugger made it easier for beginners to learn kernel hacking. Timur added, **"I lost a lot of respect for Linus when he made comments that basically implied that I was not worthy of learning the Linux kernel because I did not want to do it the hard way."** To which Marco Colombo [*] replied:

> **So Linus is right. His main argument is about people selection. Keep the ones who don't want to do it the hard way out of the door. You're just saying that his strategy is succeeding.**
>
> **Let me ask: if you want to learn how the kernel works, and you think you need a debugger, why don't you apply the patches yourself? What Linus does with his own tree has little impact on you. You won't buy Linus with the 'learning' argument, I think.**
>
> **I don't think Linus' main point is "do I the hard way or don't". The point is that "doing it with a debugger is not the easy way". It seems to be easier but you don't get the Knowledge. So in the end it's 'the right way' vs. 'the wrong way'. not 'hard' vs. 'easy'. In this, I must say I agree with him. I'd also add that source code is (should be) a language to express ideas (at the same time to implement them). With a well written source, you don't need a debugger at all to understand 'how it works'. And if you find you need it, we've quite a problem with the source in the beginning. Source code should be more human readable than debugger output, *expecially* for beginners. In this Linus is doing a great job.**
>
> **My points againts Linus' arguments:**
>
> 1. **you should give people freedom to do what they do. If they want to spend thier time with a debugger let them do. This 'social engineering' thing is crap. You don't make people better by filtering them.**
> 2. **apply tight filters on what people produce (patches, features) not on how they produce. If Linus is right (and in the end I believe it), 'debugger people' will produce low quality patches (the ones that fix the syntoms not the problems), and those patches won't be blessed by Linus. I think Alan made the point that those patches sometimes may be useful to others who are looking for the real problems.**

Frederic Magniette [*] agreed that a kernel debugger could end up letting people produce bad patches, but he added that it was very difficult to debug one's own kernel code, especially if the kernel started segfaulting. He argued, **"Then you have to begin a very slow process of printing lots of parameters and wondering about what you could have done which is so bad. This can be really awful if your code is called very often and then saturate the logs. In that way, I think that a kernel debugger could be a powerful coding tool and not only a patching tool as you say."**

There was more discussion, but at one point, Keith said, **"Various people have replied to my note on "The case for a standard kernel debugger" discussing whether or not it is a good idea. However only one person's reply matters here - Linus. I ask other people to refrain from replying to this thread until Linus has had a chance to read my note and (if he chooses) to reply to it."** Jeff replied, **"I think he did already Keith -- he said he would reject any kernel debugger submissions. :-)"**

5. Removing Distinctions Between Modules And In-Kernel Drivers

16 Sep - 23 Sep (54 posts): SCSI scanning

Jan Niehusmann reported that test9-pre1 didn't include the fix for the problem where SCSI disks were detected twice when the driver was compiled directly into the kernel (as opposed to as a module). Torben Mathiasen [*], the author of a patch to fix this, suggested adding an '#ifdef MODULE' and eventually overhauling the SCSI subsystem in 2.5; but Linus Torvalds [*] replied:

> **Please explain why it does it twice for compiled-in, and only once for modules. There must be some other logic that does the extra initializations (and that does not trigger for modules), and I'd much rather just get rid of THAT thing instead.**
>
> **I don't like it that modules behave differently from built-in. You're suggesting I make them even _more_ different, so that the end result would be similar. I'd like to know why we cannot just make them the exact same logic, and not break it more.**
>
> **Basically, there has to be some other place that does the equivalent of**
>
> ```
> if (!module) init_subsystems();
> ```
>
> **and I don't see why we can't just remove that one and make modules and compiled-in behave the same.**

Jan and Torben hunted around for a few posts, and Torben said, **"I've attached a patch that seems to do "The Right Thing". The problem was that the host detection routines would initialize the upper layers of scsi drivers (sd/st/sr), and then the module_init routines would do it again. I've removed this so**

now all device initialization is done through the module_init stuff." Linus replied, **"Looks good, and cleans up the code."** [...] **"This is the patch I was looking for. Thanks."** He rooted around in the code himself for a bit, and finally added:

> **That's another case where the SCSI layer is module dependent. If it's a module, we use the "init_module()" in scsi/scsi.c, and if not, we instead use "scsi_dev_init()". They do some of the same things (well, they obviously would have to, otherwise there wouldn't be any point to having a init routine at all), but in general do it in very different ways for no good reason I can see.**
>
> **Torben, would you mind terribly expanding on your previous patch a bit, and also cleaning this part up? As far as I can tell, we should just remove scsi_dev_init() completely, and use the module init code with an initcall(). Two less regions of #ifdef MODULE, and one less different code-path to worry about..**
>
> **Why was this done this way anyway? I've never seen this kind of setup in any of the other drivers that have been de-liced of module dependencies..**

Jeff Garzik [*] and Eric Youngdale [*] replied that this was historical code. As Eric put it, **"SCSI was made modular very early on when the modules technology was pretty primative. As time has gone on, the two initialization paths have converged, and now they are essentially redundant."** Torben said he'd started work on a patch and expected to have a preliminary version shortly. At this point Linus had second thoughts, and said, **"Actually, hold off a moment."** He explained, **"It turns out that the MODULE case does all the right things, for all the obvious reasons. I'm running a kernel that has the #ifdef MODULE stuff just removed, and it seems to be a rather easy approach. It really only required making a few things static (the init routines would clash otherwise), and removing a lot of #ifdef MODULE. (And removing some code that was enabled only for non-modules). It looks very straightforward."**

Eric asked, just for his own sense of clarity, **"What is the primary objective here - getting rid of #ifdef MODULE, or is it removing redundant code for the two paths? Or both?"** Linus replied, **"both."** and summed up the situation to that point:

> **As you probably saw, it really started out from fixing this silly bug that was introduced by mistake some time ago - which was due to both the module init and the "built-in" init code kicking in. The fact that it wasn't clear which happened where is really for me the driving force here - I'd like to avoid the same bug cropping up in half a year when somebody cleans up some low-level driver init.**
>
> **Oh, and getting rid of the init list in hosts.c is a nice bonus. It just goes away automatically if you look at the module init path instead ;)**

At one point, Linus put out test9-pre3 and a lot of people went over the change, discussing various technical points.

6. Keeping Reserve Pages Available In The New VM

17 Sep - 19 Sep (13 posts): /proc/sys/vm/freepages not writable.

Patrick Mau reported that /proc/sys/vm/freepages no longer had the write permission set in 2.4.0-test9-pre1. He asked if this was intentional, and Dave Jones [*] said that with the new VM, **"Changing this field is no longer relevant to the restructured code."** Rik van Riel [*] went into more detail, also replying to Patrick:

> **It was intentional. Writing to this file hasn't worked well since 2.3.50 or so, when the zoned VM was merged.**
>
> **Also, the fact that the new VM keeps a list of directly reclaimable inactive pages around that varies according to the amount of VM activity should make tweaking this value no longer needed...**

Andi Kleen [*] felt there should be a way to force the VM to keep lots of pages around for interrupt-intensive load such as gigabit networking. Rik gave a technical explanation that confirmed this couldn't be done, and Andi replied, **"So it cannot take load bursts. That's ok for a default, but for special loads it would be good if there was a way for the administrator to overwrite that, similar to the old freepages."** Rik suggested, **"OK, lets see if we can come up with some nice (self-tuning?) idea for this at Linux Kongress ;)"** Andi didn't like self-tuning in this case, because it would tend to drop a lot of packets on the first spike, before the VM could adjust. He said, **"When the admin says "I don't care if 10MB are wasted, I want it this way" explicitely he should get his will."** Rik agreed, and said he'd add the feature, and Andi suggested, **"It would be nice if you could do it via freepages again, then documentation would not need to be rewriten. Even if some of the numbers are meaningless now, e.g. the middle number could give a goal for the VM."**

7. PERCRAID 3 Drivers Going Open Source

17 Sep - 20 Sep (13 posts): PERCRAID 3 drivers?

Bruce A. Locke [*] got a PERCRAID 3 card from Dell that seemed only to work with a binary kernel module for kernel 2.2.14; he asked, **"A check of Dell's (rather horrible) support website only turns up the binary module mentioned above. Does anyone know anything about these PERCRAID 3 cards and if there is an opensource driver? or at least a binary module for a newer kernel?"** There were two main subthreads coming off of this. In the first, Wakko Warner [*] replied, **"AFAIK, Dell wrote these drivers themselfs and they are unwilling to release the source."** Alan Cox [*] replied, **"The drivers for the percraid have adaptec**

copyrights and have been made available finally but were too ugly at the moment to merge (and had some obvious potentially nasty bugs like using down() on a spinlock. The adaptec guys are cleaning it up." He added later, **"Im quite sure the same bug is in there binary only drivers too. I think it just happens to work for all the wrong reasons."**

In the other subthread replying to Bruce, Matt Domsch [*] from Dell said, **"The aacraid driver was submitted to Alan Cox, but rejected because it has too many "NTism's" in it, which are being addressed. Please see the Red Hat Linux "Pinstripe" beta kernel source RPM for the source code, or contact me privately."** Jon Mitchell pointed out that the sources were available from Dell's site, and at one point Bruce said, **"The aacard driver patches that were in the Redhat pinstripe kernel SRPM work fine with 2.2.17. The machine seems pretty stable and speed is about the same as with the binary driver."** Elsewhere in the subthread Igmar Palsenberg [*] remarked, **"In my opinion, manufacturers should provide docs, not some buggy peace of software that needs to be completely rewritten afterwards."** Alan replied, **"Well they are doing the rewriting so I don't mind. Its also easy to see how the code works one you read the driver. They aren't hiding stuff on the scsi side although the filesystem (yes aacraid can do its own file system on card) is missing."**

8. Getting Very Close To 2.4.0

17 Sep - 22 Sep (61 posts): Linux-2.4.0-test9-pre2

Linus Torvalds [*] announced 2.4.0-test9-pre2 and said, **"Ok. I think we're getting to the point where there are no major known bugs. That means that as of the final 2.4.0-test9 I will no longer accept any patches that don't have a critical problem (as defined by Teds list) associated with them."** Alan Cox [*] replied, **"Argh. Im not going to have time to push all the driver fixes from 2.2 into 2.4 then, I've got a house move to do yet"** And Rik van Riel [*] also replied to Linus, **"Hmm, the new VM doesn't have the out of memory killer integrated yet. I'll do some out of memory / low on swap work and will send you the patch soon, dependant on how connected I will be during Linux Kongress..."**

Russell King [*] asked about the various port maintainers, and if they'd be allowed to get their changes into the tree at this late date. Tom Rini [*] also felt this was a good question, and summed up, **"I know PPC still has some issues that need to be sorted out (We got lots of PCI resource collisions on most(all?) new machines, which really shouldn't be). I'm sure other arches have problems which haven't been posted on l-k but have been on their respective -dev list."** Martin Schwidefsky [*] added, **"I wanted to do an update for the s/390 architecture since weeks but there was always something more important. I finally cut some hours out of my ribs and made a patch against linux-2.4.0-test8. The diff for files in arch/s390, include/asm-s390 and drivers/s390 is pretty big, about 1 MB. The diffs for non s/390 files is smaller, only 35 KB. The question is now do you want to have the patch or do we wait until 2.4.1?"**

At around this point some folks started complaining about 2.4 taking too long, and Cort Dougan [*] mentioned:

> **My opinion is that 2.4 bug fixes are less common than unnecessary rewrites of working code that brings about even more instability and delays for a real and stable 2.4. Bitterness leaks out...**
>
> **I have a start to the 2.5 tree for PPC. We're keeping up-to-date with 2.4 changes so when the official 2.5 does come out we can merge our changes into it without too much trouble. I'm only allowing bugfixes into our 2.4 tree now that we have some place to play.**
>
> **If anyone else wants access to the 2.5 tree we have as a place to keep experimental changes I'm happy to open it up to the outside.**

David S. Miller [*] replied:

> **The _whole reason_ 2.5.x isn't started is so that people concentrate on stabilizing 2.4.x instead of working on new stuff. Why not just tell these people "why are you working on experimental stuff, put together PPC stress test and kernel regression suites if you are bored, because we know 2.4.x isn't read for prime time"**
>
> **You cannot complain about 2.4.x not being timely if you are doing things which directly encourage folks to not work on 2.4.x at all. Right?**

Cort replied:

> **My PPC guys want to change things. I can't stop them, but I can prevent them from screwing with 2.4 which needs to stay stable. If they don't wan to fix bugs in 2.4 then I can't force them to. They don't work for me, and I don't work for anyone who cares about PPC. So I'm not going to put a great deal of effort into forcing them to them to do something they don't want to. I've effectively kept them from making 2.4 unstable on PPC. That's a good thing, I think.**
>
> **I've also given them a place to do their experimentation before it becomes safe. Once it does, I can move it into 2.4 or wait for 2.5 if the changes don't become stable.**

David Disagreed with that approach, and the discussion went on for awhile before petering out.

9. Deadlock Hiding In New VM

19 Sep (4 posts): Rik's VM contains a deadlock somewhere

Anton Petrusevich [*] reported reported a deadlock with the new VM code on low memory systems. He'd told Rik van Riel [*] about it, but Rik hadn't been able to find the problem. He urged folks to test it thoroughly on their own systems, since **"With mem=8m it couldn't finish init scripts even."** Mike Galbraith [*] reported a similar problem on a higher memory machine (128M), and Rik said to Anton:

> **I /thought/ I had fixed this, since the system runs fine on my (SMP, SCSI) test machine when I boot it with mem=8m.**
>
> **Somebody on IRC suggested to me that this may be an UP-only bug ... I'm looking into this and hope to fix it soon, but I have to admit some help would be welcome ;)**
>
> **(I'm still at Linux Kongress and won't be back in the office for about a week)**

Anyone wanting to talk to Rik and others on IRC should check out #kernelnewbies at irc.debian.org

For coverage of Linus' acceptance of the VM patches into the official sources, see Issue #86, Section #15 (**15 Sep:** "New VM Goes In 2.4: The Saga Heats Up").

10. linux-kernel Digest Discontinued

20 Sep - 22 Sep (14 posts): linux-kernel-digest ?

Robert Greimel asked if the linux-kernel digest would be coming back (it was left out when linux-kernel moved from vger.rutgers.edu to vger.kernel.org), and Matti Aarnio [*] replied that according to David S. Miller [*], no. He went on, **"Sometimes he holds possibly bad opinnions as ferociously as Linus, but on the other hand, that Majordomo 1.x digester breaks structured MIME messages BADLY. It should be trivial to fix, but I don't hack Md, I hack ZMailer -- and also sometimes the kernel."** Dmitry Pogosyan [*] replied in dismay, **"This is very unfortunate, since linux-kernel-digest was the great way to keep many interested people informed about state of Linux development Without it, > 200 mails daily in linux-kernel is fairly prohibitive if you are not an active developer."** Matti suggested using 'procmail' to sift linux-kernel into its own mailbox.

Elsewhere, Alan Cox [*] remarked, **"Having slowly switched every list I have to run or help run to mailman I can recommend the pain of the switch over the pain of running majordomo."** David replied, **"I suggested this but Matti immediately told me that mailman has several RFC compliancy "issues" which I am sure he can elaborate on."** Alan expressed interest in hearing more, and Matti replied:

> **A year or two ago I had a peek after I got reports that it didn't work**

> **together with ZMailer -- turned out that some baseline PYTHON library for SMTP does something stupid in its protocol line construction.. The system experiencing problems was running ZMailer with ultra-strict RFC 821 (plus amendaments) mode.**
>
> **Ergo, the MAILMAN might be ok, but underneath it there may be buggy language library.. (may STILL be, that is.)**
>
> **Another point, DaveM, was that MAILMAN is written in Python, and neither of us "speaks" that language...**

Oliver Xymoron [*] remarked, **"Python is quite readable though. Hacking existing code isn't too painful for anyone who knows Perl and C."** No one replied to that, but elsewhere, Steven Pritchard [*] mentioned:

> **There is a Majordomo 2 in development. I'm using it for all of my lists now. The upgrade from Majordomo 1 wasn't terribly painful.**
>
> **Note that it's not really a new version, but rather a total rewrite. MIME, archives, digests, and everything else you would expect to "just work" does. Of course, the catch is that there hasn't been an official, stable release yet.**

It seems that for linux-kernel, folks wanting the old digest will have to wait for the majordomo upgrade or the changeover to different software.

11. New VM May Not Make It Into 2.4

21 Sep - 22 Sep (22 posts): [patch *] VM deadlock fix

Rik van Riel [*] announced:

> **I've found and fixed the deadlocks in the new VM. They turned out to be single-cpu only bugs, which explains why they didn't crash my SMP tesnt box ;)**
>
> **They have to do with the fact that processes schedule away while holding IO locks after waking up kswapd. At that point kswapd spends its time spinning on the IO locks and single-cpu systems will die...**
>
> **Due to bad connectivity I'm not attaching this patch but have only put it online on my home page:**
>
> **http://www.surriel.com/patches/2.4.0-t9p2-vmpatch (http://www.surriel.com/patches/2.4.0-t9p2-vmpatch)**
>
> **(yes, I'm at a conference now ... the worst beating this patch has**

> **had is a full night in 'make bzImage' with mem=8m)**

A lot of folks reported the same or worse deadlocks, and at one point Linus Torvalds [*] warned, **"those VM patches are going away RSN if these issues do not get fixed. I'm really disappointed, and suspect that it would be easier to go back to the old VM with just page aging added, not your new code that seems to be full of deadlocks everywhere."** Rik replied:

> **I've been away on a conference last week, so I haven't had much chance to take a look at the code after you integrated it and the test base got increased ;(**
>
> **One thing I discovered are some UP-only deadlocks and the page ping-pong thing, which I am fixing right now.**
>
> **If I had a choice, I'd have chosen /next/ week as the time to integrate the code ... doing this while I'm away at a conference was really inconvenient ;)**
>
> **I'm looking into the email backlog and the bug reports right now (today, tuesday and wednesday I'm at /another/ conferenc and thursday will be the next opportunity).**
>
> **It looks like ther are no fundamental issues left, just a bunch of small thinkos that can be fixed in a (few?) week(s).**

A little bit before this exchange, Rik found a spot in the code where he'd inadvertantly reversed the logic on a test in an 'if' statement. Ingo Molnar [*] tested it out right away and replied (with a patch), **"yep this has done the trick, the deadlock is gone. I've attached the full VM-fixes patch (this fix included) against vanilla test9-pre5."** Rik asked Linus to include Ingo's patch in the next pre-release, but André Dahlqvist and Yuri Pudgorodsky reported oopses with the patch. There was no reply, and the thread ended.

Elsewhere, under the Subject: test9-pre6, Linus Torvalds released 2.4.0-test9-pre6 and remarked, **"This should fix the VM deadlocks (knock wood)."**

12. es371 Sound Card Catches Bad Fixes

22 Sep - 24 Sep (12 posts): No sound (es1371) after test7

Someone reported that their es1371 sound card stopped working in 2.4.0-test9-pre5. The last kernel he'd tried had been test7, which had worked fine. Mordechai Ovits added that the same card worked fine for him under test8. Linus Torvalds [*] looked around in the code and found a bugfix that seemed to miss the point. He suggested reverting it, but then replied to himself later with another bogus bugfix. Jon Evans reported success with Linus' fix, and Linus said, **"I will now hunt down the person who sent me that patch, and do nasty things to him. After I whip myself for accepting it in the first place."** Alan Cox [*] replied that he'd add the fix to 2.2.18pre, and added, **"Its an escapee bug from 2.2 that leaked into 2.4. Not the fault of whoever forward ported, its a debugging thing that wasnt removed."** Jeff Garzik [*] took the bullet (also in reply to Linus), with, **"hmmmm The patch was submitted by me, but it came straight from 2.2.x... after being whipped, I shall look into both versions of ac97_codec some more..."** Linus remarked, **"I'm surprised that it works at all in 2.2.x - that implies that the 2.2.x code is completely different from the 2.4.x logic."**

We Hope You Enjoy Kernel Traffic