Kernel Traffic #84 For 11 Sep

By Zack Brown

Table Of Contents

Introduction

I'd like to draw folks' attention to Kernel Cousin Debian, the newest Cousin, following the debian-devel mailing list. This is a group-authored Cousin, and is dependent on volunteers. If you've got 1/2 hour per week and are interested in helping, check out the KC Debian home page (http://kt.linuxcare.com/debian) or just read issue #1

(http://kt.linuxcare.com/debian/dd20000907_1.html).

This week, thanks go to Peter Samuelson (Hi Peter!) for snagging an HTML error in the last issue. An <li> without the >... Thanks, Peter!

Matthias Andree also had some additional information for Issue #83, Section #3 (**20 Aug:** "Dealing With Binary Files In The Kernel Source"). He wrote to me:

> **In KT #83's thread about encoding binary files in the kernel source tree, I have received some personal mail, one piece therein that pointed out that the uuencode documentation was available in the Single Unix Specification, either for US$290 or online at http://www.opengroup.org/ (http://www.opengroup.org/)**
>
> **I have been given these URLs that point directly into the documentation and can confirm them working.**
>
> **http://www.opengroup.org/onlinepubs/007908799/ (http://www.opengroup.org/onlinepubs/007908799/) http://www.opengroup.org/onlinepubs/007908799/xcu/uuencode.html (http://www.opengroup.org/onlinepubs/007908799/xcu/uuencode.html)**

Thanks for the pointers, Matthias.

Thanks *again* go to Frederic Stark for catching me napping on the links into the archives again. They're back this week, though note that not all of them go directly to the start of their thread. This appears to be fallout from the switch from rutgers.edu to kernel.org, that will hopefully not be a problem next week. Thanks, Frederic!

Marko Kreen also sent me some mail about the "eliteness" of kernel hackers, as discussed in last week's introduction. He pointed out that the #kernelnewbies channel of irc.openprojects.org is a great IRC channel for talking about kernel issues. It's friendly to newcomers but can get as technical as you want. There is also the corresponding web site (http://kernelnewbies.org/) and even a mailing list at kernelnewbies@nl.linux.org. (To join, send "subscribe kernelnewbies" in the body of an email to majordomo@nl.linux.org. Thanks, Marko! I should have included all this information last week without prompting, since I go to #kernelnewbies myself and find the folks there very helpful and fun to talk to. Good catch!

Also on the subject of linux-kernel's eliteness, Dave Madsen wrote to me to say:

> **I, too, find the list to be exclusive territory.**
>
> **First, I posted a bug report that I thought that someone might be interested in. I won't go into details, but there was NO response in any way. Yes, it's a kernel issue.**
>
> **Second, the Linux kernel has become complicated enough that there's significant barriers to entry to hacking it in a meaningful way. Because of the relatively limited number of people really contributing to the project (as can be seen by the number of posters), it seems to be that the kernel is being run more as a closed project with source available than an "open-source" project. (Yes, I can participate as a victim/tester :-), but I can do the same thing with Billy-boy's stuff, too). Linus is, of course, "project manager" -- he gets to say "NO! THOU SHALT NOT!".**
>
> **It would be interesting to see ESR analyze what happens to an open source project as the barrier to entry increases. I don't recall that his papers deal with this issue (but RTFM references welcome).**
>
> **Sure, go ahead, point me to all the sites with various patches. I believe that because these patches aren't in genuine "official" kernel, they aren't really meaningful. How many distros include these patches? Let's face it, most folks don't even really look for any "3rd-party" patches.**
>
> **I would also like to see some significant corporate contributions, also -- and I don't mean a Linux**

> **company. How about getting the proper techniques for parallelism from Dynix (Sequent/IBM's NumaQ)? It's been in there a long time. I think IBM's doing that -- I hope they finish soon!**
>
> **Please don't get me wrong; I've used and loved Unix since 1986, and was there when all the Unix magazines and papers started covering NT because it looked like NT was going to take over the world. I'm a Linux enthusiast, and don't hesitate to promote it every chance I get.**
>
> **But I also face the facts -- the Linux kernel is closed unless you've spent SIGNIFICANT time trying to learn and hack it. The list members don't say this, but they know it; they're proud of the efforts they've put into learning it to be able to do what they do -- and they should be. But, as a rule, they DO ignore outsiders.**
>
> **Please don't consider me to be placing blame or passing judgement; things are the way they are, and that's it. But to deny the facts is foolish.**

Well, the first thing I would say in reply to this is that it is quite a valid perception, and a very important one to express, since it relates directly to the fundamental nature of the Linux development process. Maybe these things should be discussed on linux-kernel, and see what folks have to say about it.

But I still disagree. First of all, regarding your unanswered report, this has nothing to do with your being a newcomer or an oldtimer. Not all posts get replies, no matter who they're from. You're not the first person to complain about that, and I'll give you the same advice I've seen given to other folks: if you post a report to linux-kernel that doesn't get a reply, post it again. If you say, "this is the Nth time I'm posting this report", people may try harder to reply to it. But getting frustrated and giving up won't help anybody do anything. Remember, it's a big mailing list, a very *very* big mailing list. Even if your report is important, it may still get lost in the shuffle.

Now, the idea that the barrier to entry is growing, because the code is becoming more difficult and complex, seems like a non-objection. What is the barrier to working on the MS Windows kernel? Now there's a barrier. Linux has no barriers of that kind. The barrier is only yourself. You have the code. You have the brain. Start hacking. Kernel hacking is not a dark art. Nor is it tic-tac-toe. It's not for everybody. But it *is* for *any*body.

In any case, thanks for the comments, Dave. I think linux-kernel is a better place for future discussion of this issue though. Hopefully that post will get a reply. ;-)

Mailing List Stats For This Week

We looked at 983 posts in 4191K.

There were 317 different contributors. 150 (47%) posted more than once. 114 (35%) posted last week too.

The top posters of the week were:

- 56 posts in 162K by Alan Cox <alan@lxorguk.ukuu.org.uk>
- 39 posts in 145K by Linus Torvalds <torvalds@transmeta.com>
- 34 posts in 121K by yodaiken@fsmlabs.com
- 32 posts in 145K by Alexander Viro <viro@math.psu.edu>
- 31 posts in 110K by "Andi Kleen" <ak@suse.de>

1. Posix Threads (pthreads) In Linux

23 Aug - 1 Sep (208 posts): SCO: "thread creation is about a thousand times faster than on native Linux"

The Claim And The Way It Is

Joerg Pommnitz gave a link to an article (http://www.theregister.co.uk/content/1/12733.html) in The Register that claimed thread creation under SCO was about a thousand times faster than under native Linux. Very little of the discussion that followed directly addressed the article, but Ingo Molnar [*] closed the vice immediately with some numbers:

> **Linux creates a 'native Linux thread' via clone() in about 10 microseconds. So SCO creates a thread in 10 nanoseconds - ie. faster than a cacheline load on most CPUs - wow!**
>
> **glibc creates a pthread in about 300 microseconds, thats certainly slow. (but then again, still faster than NT's thread creation latency.) I doubt SCO creates a pthread in 0.3 microseconds.**

Jeff V. Merkey [*] also addressed the article, saying, **"This whole SCO thing is overrated. I worked on the Unixware code base at Novell, and it's putrid in comparison to Linux. It's got a lot of good apps, but so does Linux. This kind of tripe propaganda is what the "cult" followers of Unixware are good at. They lost @ $ 38,000,000 dollars each year at Novell ramming UnixWare down our throats and pissing of our customers -- Novell finally cut rheir losses and sold it. Don't get me wrong, it's decent Unix stuff, but Linux is tons better as a general purpose PC Unix."**

Tigran Aivazian [*] wondered aloud whether any commercial UNIX had anything at all that Linux could benefit from. Offhand, he couldn't think of anything, though he'd been a UnixWare7 escalations engineer for 2 years. Andi Kleen [*] replied, **"How about really working SCSI error handling ? @)."** And Tigran corrected himself as well - he *could* think of something from UnixWare7 that Linux could use: the VERITAS vxfs journalling filesystem. But Alan Cox [*] replied, **"When it becomes open source maybe. Until then I think XFS, Reiserfs etc over LVM and software raid look way more appealing."**

Elsewhere, Tigran also considered, **"It is plausible that individual _lwp_create(2) is faster than individual clone (CLONE_VM) one (possibly between 5x and 10x times, the 1000x figure is ridiculous). However, one should look a bit further and ask if native kernel-level UW7 lwps are delivering the same set of rich functionality as Linux clone'd tasks. IMHO, they simply don't. So one has to add all the overhead of turning a "raw" lwp into a useful and useable thread, e.g. as used by userspace thread concept. If you do that, no doubt you will find that Linux**

performance is greater than of any commercial alternatives." Andi Kleen remarked, **"The main problem Linux clone has over LWPs is the extensive memory resource usage (~8.5K on x86, 16+somethingK on 64bit for the kernel stacks). To solve that it would be very nice to have Mach like continuations in Linux (multiple kernel threads sharing a kernel stack)"** Linus Torvalds [*] replied:

> **Oh, Gods, no.**
>
> **Continuations are evil. And they don't scale in SMP. Don't even think about them.**
>
> **Mach had a lot of bad ideas. In fact, I can't think of a single _good_ idea from Mach. This one certainly was not.**
>
> **(As far as I can tell, the only reason for continuations in Mach was that Mach performance sucks without them, and it was a way of handling the process switches from a client process into a server process without the overhead. Of course, the right solution to that particular problem is to not use a microkernel at all, but that was a bit too non-radical for the Mach guys).**

Later on in the same subthread they came back to the memory resource usage Andi had mentioned earlier. Alan said in the course of discussion, **"The kernel has fast lightweight threading,"** and Andi replied, **"I would not call 8K/16K overhead "lightweight" treading."** Linus replied:

> **Oh, and pray tell how much you expect a kernel thread to take?**
>
> **The memory overhead is negligible. An app that thinks that 16kB/thread is too much, should not be using kernel threads in the first place. You could argue that it probably shouldn't be using threads at _all_.**

Jamie Lokier [*] pointed out some optimizations that could be made, such as removing the stack page from sleeping-but-runnable tasks. But Linus replied:

Wrong.

A stack page _is_ required for a thread, basically always:

1. **when it is running (either in user or kernel space): interrupts happen. You can't share the stack page sanely with other processes, because the interrupt can (and often does) cause a task-switch to another completely unrelated process. And you have to save the register state somewhere.**

   NOTE! Linux does NOT save the register state in the task struct. Saving state in the task struct is a horrible waste of time when it has been already saved on the stack for other reasons (ie the stack contains not just the register state, but the instructions about how to unwind it from recursive faults etc).

2. **when it is sleeping in kernel space: the stack contains the full path to where it slept, we _need_ it.**

Basically, the _only_ time you don't need a kernel stack is when the process is completely idle, ie it has been scheduled away because its time-slice ended, and it wasn't in kernel mode when this happened. And Linux doesn't consider this a special state at all - as far as the kernel is concerned, that sleeping state is the same as sleeping in kernel mode.

Quite frankly, the stack _is_ part of the "struct task_struct". It's not just a performance optimization to allocate them together. Sure, the fact that we allocate them together means one less allocation, and it allows the clever "current" games we play with the stack pointer, but those are details. Fundamentally, the two are part of the same thing.

Yes, you could do it differently. In fact, other OS's

> **_do_ do it differently. But take a look at our process timings (ignoring wild claims by disgruntled SCO employees that can probably be filed under "L" for "Lies, Outright"). Think about WHY our system call latency beats everybody else on the planet. Think about WHY Linux is fast. It's because it's designed right.**
>
> **Continuations and other crap only add overhead. 8kB of storage for the whole thread state (including the kernel stack) is _incredibly_ tiny.**

He finished, **"Trust me. You won't find faster true threads than what Linux offers. And they won't be using less than 8kB of memory."** Jamie disagreed, and gave a number of technical reasons why some optimizations were possible, and Linus replied, **"Talk is cheap. Show me the code."** Later in the subthread he elaborated:

> **I think what Jamie is overlooking is all the details.**
>
> **For example, take the really obvious one: sure, you don't need a kernel stack under some circumstances.**
>
> **What are you going to do when you DO need one?**
>
> **Allocate one dynamically?**
>
> **Good-bye performance.**
>
> **That's why I said "Show me the code". I'm rather unconvinced that you can actually get the details right without huge amounts of complexity.**
>
> **I'm sure it could be done. I'm pretty convinced that there is no way to do it efficiently.**

Historical Digression

Elsewhere (and throughout most subthreads) various implementation details came under discussion; and at one point in response to a possible set-up, Albert D. Cahalan [*] bemoaned:

> **I hate this.**
>
> **First of all, I dare not code it. Linus seems to want Linux not strongly tied to the POSIX thread model.**

To which Linus replied:

> **Stop this bandying my name about.**
>
> **The only thing Linus wants is SANE SEMANTICS.**
>
> **It so happens, that POSIX seems to be insane. Quite frankly, some of POSIX is utter crap. Cow-dung. In short, shit.**
>
> **And the reason pthreads have some problems is because the stupid thing was never designed. It grew out of 1:m implementations and the Sun threading code.**
>
> **I want to have pthreads support for Linux. But I do not want the crap in pthreads to get into the kernel. Which means that I want people to THINK before they code something up. A 1:1 pthreads implementation simply isn't going to make it, but something that happens to have the braindamaged pthreads stuff as a subcase (with some help from user space) is more than welcome.**
>
> **This is not something new. I, unlike the POSIX committee, have more taste than willingness to get reamed up the ass by years of history.**
>
> **Deal with it.**

Marty Fouts [*] replied:

> **With all due respect, as a person who was present for the early evolution of pthreads, I have to disagree with your assertion of how pthreads came to be.**
>
> **Pthreads was designed. The initial proposal that**

was presented to P1003.4 was based on a fully functioning multithreaded programming model already in use (at DEC SRC, IIRC.) Unfortunately, threads and Un*x don't mix without breaking something. P1003.4 decided to break the threads model in order to preserve Unix semantics. The design was fine; if there was a problem it was with the evolution and the compromises needed.

None of this matters to the fact that, eventually, if one wants C/Un*x semantics and pthreads (or any multithreaded model) and one want smp scalability beyond a trivial number of processors, one will have to both cope with BAD (broken as designed) semantics, and with kernel/library interaction to resolve performance problems, especially in synchronization.

Grafting 'thread' semantics onto a programming language that wasn't designed for them (C) and an operating system that had a large body of semantics that weren't easily made compatible with them (Un*x) left us needing a compromise that wasn't perfect. That is, after all, what engineering is about: making the compromises necessary to solve the problem at hand with the tools available.

After all, as Eric Serveraid was fond of saying "the principle cause of problems is solutions."

While you are contemplating how to extend and generalize clone operations, may I suggest checking the literature for "variable weight process" (an idea I seem to recall being invented at Sequent 15 years ago,) and Brian Bershad's papers on scheduler activiations?

And don't worry about history. It will come back to haunt you in 10 years.

Linus replied conciliatorily:

Thanks for the corrections. And I'm sorry if I

> **stepped on any toes.**
>
> **However, I still would not call "pthreads" designed.**
>
> **Engineered. Even well done for what it tried to do. But not "Designed".**
>
> **This is like VMS. It was good, solid, engineering. Design? Who needs design? It _worked_.**
>
> **But that's not how UNIX is or should be. There was more than just engineering in UNIX. There was Design with a capital "D". Notions of "process" vs "file", and things that transcend pure engineering. Minimalism.**
>
> **In the end, it comes down to aesthetics. pthreads is "let's solve a problem". But it's not answering the big questions in the universe. It's not asking itself "what is the underlying _meaning_ of threads?". "What is the big picture?".**
>
> **That doesn't mean that there wasn't a lot of worthwhile work put into it. So sorry for some of the more colorful commentary..**

Dave McCracken [*] also replied to Marty:

> **You are in fact right. The original model for Pthreads came from DEC SRC and its Topaz system (written entirely in Modula 3). In fact, the CMU cthreads library was written by a student who spent a summer internship at DEC SRC prior to writing it.**
>
> **The proposal was submitted to P1003.4 by way of Apollo, where we tried to make it somewhat compatible with Unix semantics (when Garrett Swart of DEC SRC had Unix signals explained to him he was horrified :). The fundamental paradigm we tried to preserve was the concept of a process as the fundamental building block of an application (where a process was defined as an invoked program running in an address space),**

> **and threads were simply a way to add some concurrency within that process context. We deliberately avoided the concept that threads have any existence other than as part of that process and its address space.**
>
> **Linux is clearly starting from an entirely different paradigm in its clone() semantics. While this may well be a better model, I don't think it's fair to call POSIX threads broken and stupid because it wasn't written for the Linux way. Linux's definition of a thread is clearly different than we used when pthreads was designed, but that doesn't make pthreads 'wrong' or 'crap' or 'shit' as Linus is so fond of saying. I'll agree that it makes it difficult to reconcile the two semantics and implement a pthreads library on Linux.**
>
> **At any rate, I think it should be possible for us all to agree that Linux threads and pthreads have significant differences without resorting to the kind of name-calling I've been seeing in this discussion. Much thought by some very smart developers went into the pthreads design, and how to make it compatible with the Unix semantics we had to start from. The world and the idea of a thread may have evolved beyond what pthreads tried to address, but that doesn't mean it's necessary to heap derision on it.**

Victor Yodaiken [*] replied, **"What is "broken" about Pthreads is the clumsy interaction between UNIX semantics and threads and it seems to me that this is all due to overdesign and some false assumptions in the threaded programming model that was current 15 years ago. While we do have the benefit of hindsight, the same overdesign continues in POSIX drafts."** He went on, **"One of the reasons I like Linux is that nobody has too much reverence for past design decisions. In fact, a famous Australian Linux developer recently described his own code as "20,000 lines of untested crap", but I think that was rather unkind as there are somewhat less than 20,000 lines."** Dave acknowledged:

> **Ok, I'll grant you that some parts of it were crap, and some crap was added in later drafts. I'll admit I've lost touch with the details that've been tacked on since about Draft 4 or so.**
>
> **I'll also admit signals were always a mess. They never fit very well with pthreads, and we pretty much punted in the early drafts. I added sigwait() to the original draft as a way an application could direct all its signal handling to a single synchronous 'signal handler' thread, but it never addressed the larger question of how to make signals and threads play together.**

But he added, **"All of POSIX was a good example of what 'design by committee' really means, especially when various members of those committees worked for competing companies with their own agendas. But on the whole I think we came up with a decent basic design for what it was, as someone earlier put it... 'thread as a component of a process' as opposed to Linux's 'thread as a process'."**

But Marty also replied to Victor:

> **I'm curious: what assumptions did we have about threads 15 years ago that turned out to be false? What I recall from attending the P1003.4 meetings during the first few years of the pthread debate was that there were three different communities wanting to use 'threads' of control for three different reasons. The original proposal came from people who were interested in threads as a programming abstraction for making certain kinds of programs easier to write; the real time community had a completely different take, wanting 'extremely lightweight' processes; and the parallel programming community was looking at various concurrency models.**
>
> **On the other topic: I took part in interminable debates about the interaction between threads and various Unix facilities, including the wrangling**

> **over signals. The reality is that (as confirmed later in discussions with Ritchie,) signals in Un*x are "broken" in that they were never intended to be used as an IPC mechanism other than at a gross level, mostly for precise interrupts caused by program failure. The problem was that they had come to be used as an IPC mechanism and at that time there were (at least) two competing proposals to 'fix' signals.**
>
> **In the committee we were stuck trying to resolve a battle between a community that wanted per-thread signals, a different community that wanted threads to not (necessarily) be visible outside of the calling process, et cetera. The answer to your 'why not' was that we deliberately avoided distinguishing between thread and process precisely to allow both library-only thread implementations and thread=process implementations. At a time when 'session' and 'job' were being added to Unix semantics to allow terminate of related groups of processes (for administration of large systems) there simply wans't a way we could put together a definition like yours that wouldn't have caused the community that liked 'the other' model to vote against the spec.**
>
> **The *real* problem with posix wasn't overdesign, but rather, overgeneralization. When you are doing an extention to an OS standard that doesn't even specify the memory model, it is hard to not overgeneralize.**
>
> **Standards committes aren't in the business of pointing out the one true path; they are in the business of describing the least offensive compromise among existing practicies and pet theories.**

Linus replied:

> **Personal opinion, and it has nothing to do with "15 years ago" vs "today":**

The pthreads approach never got to a real framework for threads as real entities. To pthreads, a thread is a braindamaged stepchild of a process, and cannot do anything on its own. It's this drooling messy thing that has no life without the parent process that wipes up after it. It has no spine.

In short, threads are not proper citizens. They are guest workers. Expendable. Worthless. They don't have a life of their own.

Now, the notion of rfork/clone/sproc "variable-weight processes" is not new per se. But it's an important _notion_. It basically says that threads are _not_ the ugly drooling stepson that you really wouldn't want to see at family re-unions.

Suddenly, with rfork/clone/sproc, a thread is not just something that you can prod in the right direction with the cattle prod of a random collection of POSIX routines. A thread is an Idea. A Notion. Something worthy of a capital letter. Something you can discuss in mixed company.

It's the difference between being useful and being Right.

It's hard to explain. If you have a bent toward physics, it's the difference between a practical experiment and the Unified Teory of Everything. It's the difference between Galileo saying "everything falls at the same speed" and Newton's "F = mMG/r?".

If you're religious, it's the difference between "It was a dark and stormy night.." vs "Let there be Light!".

If you're into computers, it's the difference between Windows ("sure, it works much of the time, and it looks pretty") and Unix ("everything is a process or a file").

> **It's like an idea: there are mundane ideas ("hey, let's go out for pizza and a beer") and there are big ideas ("I have a dream..").**
>
> **The difference? The big idea leads to something larger than itself. It makes people think about what the meaning of life is. It gives a _direction_ for where things are supposed to go.**
>
> **In contrast, a small idea leads to other small ideas (fifteen beers later: "I know, let's drive past the police station and moon every cop in the city!").**
>
> **Ok. I'm overdoing it. But think of rfork/clone/sproc as a way to try to come to grips with what it really means to be a thread. Be one with the thread. Grok the threadedness - so that you can understand what is wrong and what is right _without_ having to count every comma in a standards draft that is 473 pages long.**
>
> **In short, "pthreads" is a rough approximation of the theory of magnetism. While rfork/clone/sproc is Maxwell's equations. One can tell you how much force a magnet excerts on a charged particle in motion. The other one tries to explain how the universe works.**

Marty had the last word, with:

> **Pthreads originated with the community that cared about thread-as-programming-extraction, and not thread-as-operating-system-concept. It was stuck with a need to support the lowest common denominator, ie, library-threads-in-a-process-on-a-uniprocessor. No matter what else was done, it had to have semantics that made sense in that case. (After all, some would say, even at that time, if you want processes to share memory, you can do that with SYSV memory ops.)**
>
> **Since I do have a physics bent (my only formal training is as a mathematical physicist, after all,) I think I'd draw a different analogy. pthreads are a**

programming language abstraction, not an operating systems abstraction. They are, ie, physical chemistry, when you keep wanting to do particle physics. (and if you think your theory of magnetism/maxwell model holds, I have some reading in the theory of concurrent computing to recommend -- even 'pure threads' are primitive by comparison to, say, Chandy and Misra.)

But if you want 'Right', you are in the wrong business. Operating systems are about blurring the boundary between programming language and 'virtual machine' abstractions and real devices. In my opinion, DMR was probably the last person who got a shot at designing both a language and an OS at once, and producing something economically viable, for a very long time into the future. (There are folks at DEC who got to design both, but Modula3 is not going to replace C/C++... and ML is a better 'Right' language, anyway.) Once you've adopted C, you are stuck with engineering and not abstraction. Besides, not all computers, even general purpose computers, are used for the same thing, and it is the rare algorithm that stretchs across the entire application domain, even in threading.

My opinion is that you will end up with a split level thread design in Linux, simply because of the synchronization/scheduling interaction. Once you've accepted that, you'll have to struggle with the three kinds of thread: threads-as-programming-abstraction, threads-as-concurrency-mechanism[with realtime overtones] and threads-as-multiprocessing-mechanism. Based on experience with a range of concurrency models spanning more years than I'd care to admit to, I suspect that you'll end up with a model like variable-weight-processes (which clone() isn't that far from, now,) with Bershad's scheduler activations or something similar.

We're really not talking about Right/wrong or 'big ideas' and 'little ideas.' (but i'll be glad to buy you

> **a beer anytime you're in Mountain View.) n:m threads versus 1:1 threads is pretty much like comparing different superstring theories when you don't even have a good estimate of the mass of the universe. You want a big idea: throw the process/thread model away and do something like Arvind's dataflow machines, or design an operating system to support declarative rather than imperative programming.**
>
> **And you'll find the whole thing will be an exercise in performance tradeoffs between threads-as-programming-language constructs and scalablility in multiprocessor applications. (oh, and synchronization primitives are a critical part of a thread model . . . -- that's one of the reason's Java's is so broken.)**
>
> **But it won't get really interesting until the processor count is O(64) and the system is NUMA. ;)**

New Ideas

Elsewhere, in reply to the very first post from Joerg's, the discussion of actual Linux implementation continued, when Albert D. Cahalan said:

> **Ulrich Drepper has repeatedly flamed Linus for not adding the features needed for low-overhead POSIX threads. Linus thinks the POSIX thread interface is broken, so he won't add the ugly bits needed to sanely reach POSIX compliance. Ulrich has tried to write a fully-compliant POSIX thread API, always choosing correctness over performance.**
>
> **Thanks to this fight, we get a severely bloated almost-POSIX monster that damn near everyone hates. The lack of a unified source tree (as found in the BSD world) only makes this worse. There is no global tree owner to resolve this dispute.**
>
> **To POSIX or not to POSIX, that is the quarrel. Too**

> **bad that it flushes our performance down the toilet. We've seen this before and will see it again: sysctl, floating-point initialization...**

Linus objected, **"Wrong. I'd be happy to add the bits, it's just that nobody has sent me a sane patch. People complain a lot, but I haven't seen anything constructive."** And Mark Kettenis [*] added, **"Linus is right. There just seems to be nobody with the right set of skills who is interested in doing the work, so it just doesn't get done."** Theodore Y. Ts'o [*] went on, **"I'll echo this. In the Linux Standards Base group, it was hard to find *anyone* who was actually psyched about Posix Threads. Even most of the database vendors we talked didn't care --- they had their own multithreading solutions that was either fine with the existing Linuxthreads, or emulated it using processes and shared memory. According to one db vendor we talked to, there are enough incompatibilities in other OS's thread implementations that they don't like to depend on POSIX threads at all."**

But Albert Continued in reply to Linus, **"Nobody will send you a sane patch without you at least hinting at what you might like to see. I'm sure many of us would be happy to write the code, but not under the expectation that it will be rejected. I wasted quite a bit of my time on the last patch I sent you, and I'm not about to do that again."** Linus replied:

> **Acceptable solution:**
>
> 1. **add "tgid" (thread group ID) to "struct task_struct"**
> 2. **CLONE_PID will leave tgid unchanged.**
> 3. **non-CLONE_PID will set "tgid" to the same as pid**
> 4. **get_pid() checks that the new pid is not the tgid of any process.**
>
> **Basically, the above creates a new "session pid" for the collection of threads. This is nothing new: the above is basially _exactly_ the same as p->pgrp and p->session, so it fits quite well into the whole pid notion.**

> **It also means that "current->pid" basically becomes a traditional "thread ID", while "current->tgid" effectively becomes what pthreads calls a "pid". Except Linux does it the right way around, ie the same way we've done sessions and process groups. Because, after all, this _is_ just a process group extension.**
>
> **Now, once you have a "tgid" for each process, you can add system calls to**
>
> 1. **sys_gettgid(): get the thread ID**
> 2. **sys_tgkill(): do a pthreads-like "send signal to thread group" (or extend on the current sys_kill())**
>
> **Now, the problem is that the thread group kill thing for true POSIX threads signal behaviour probably has to do some strange magic to get the pthreads signal semantics right. I don't even know the exact details here, so somebody who _really_ knows pthreads needs to look long and hard at this (efficiency here may require that we have a circular list of each "thread ID group" - ie that we add the proper process pointer list that gets updated at fork() and exit() so that we can easily walk every process in the process group list).**
>
> **Discussion welcome. Basically, it needs somebody who knows pthreads well, but actually has good taste despite that fact. Such people seem to be in short supply ;)**

Alan, Andi, Victor, Mark, Albert, Jamie, Alon Ziv, Mitchell Blank Jr [*], Horst von Brand [*], Alexander Viro [*], Jesse Pollard [*], Kai Henningsen [*], Erik McKee, Marco Colombo [*], Pavel Machek [*], David Howells [*], David Wragg [*], and Stephen C. Tweedie [*] all discussed the implementation details with Linus. At one point Linus remarked, **"What _I_ want when it comes to threading is to have the "sensible structure" in place. I then, as a second-order interest, want to make it as straightforward and clean as possible (note the "as possible") to implement a pthreads layer on top of that,**

despite the fact that I don't particularly like the pthreads model. (At this point, a big "Oh, REALLY?" is heard across the land. Duh.)" In response to some particular implementation detail from Alexander, he also added, **"Not unreasonable. Anyway, this is all 2.5.x.. I'd like to get at least some basic support for pthreads compatibility if we can do it trivially in 2.4.x, but these kinds of changes are not necessary for that at this point."**

2. Philosophy Of Coding Style

27 Aug - 31 Aug (11 posts): [PATCH] af_rose.c: s/suser/capable/ + micro cleanups

Arnaldo Carvalho de Melo [*] posted a patch that, among other things, transformed a 3-line 'if()' block into a 1-line '?:' statement. Specifically, he changed

```
if (copy_to_user(optval, &val, len))

    return -EFAULT;

return 0;
```

into

```
return copy_to_user(optval, &val, len) ? -EFAULT : 0;
```

Kenneth Johansson felt the old way was more readable, and Philipp Rumpf [*] agreed. But Albert D. Cahalan [*] said:

> **The new way is faster to read and understand when you consider that this code isn't all alone. With the new way, my editor can display 300% more code. This reduces screen-to-brain latency by reducing the need to scroll.**
>
> **Scrolling kills productivity.**
>
> **I'd really rant if I still used a VT100 or the Linux console.**
>
> **The new way better even if it didn't save space. It clearly expresses that _something_, to be chosen based on the copy function, will be returned. This isn't a case of one code path that returns and one**

> **that does lots of other stuff. Hey, be glad the ?: operator didn't get nested a few times, maybe with the comma operator thrown in for extra fun. You just have to get used to it.**

Elsewhere, Rogier Wolff [*] said:

> **The kernel is a multi-million-lines-of-code piece of software. Software maintenance cost is found to correlate strongly with the number of lines-of-code.**
>
> **So, I would prefer the shorter version.**
>
> **If it takes you a few seconds to look this over, that's fine. Even it the one "complicated" line take twice as long (per line) as the original 4 lines, then it's a win.**

But Linus Torvalds [*] replied:

> **I disagree.**
>
> **Number of lines is irrelevant.**
>
> **The _complexity_ of lines counts.**
>
> **And ?: is a complex construct, that is not always visually very easy to parse because of the "non-local" behaviour.**
>
> **That is not saying that I think you shouldn't use ?: at all. It's a wonderful construct in many ways, and I use it all the time myself. But I actually prefer**
>
> ```
> if (complex_test)
>
>         return complex_expression1;
>
> return complex_expression2;
> ```
>
> **over**
>
> ```
> return (complex_test) ?
> complex_expression1 :
> ```

```
        complex_expression2;
```

because by the time you have a complex ?: thing it's just not very readable any more.

Basically, dense lines are bad. And ?: can make for code that ends up "too dense".

More specific example: I think

```
        return copy_to_user(dst, src, size) ? -
        EFAULT : 0;
```

is fine and quite readable. Fits on a simple line.

However, it's getting iffy when it becomes something like

```
        return copy_to_user(buf, page_address
        (page) + offset, size) ? -EFAULT: 0;
```

for example. The "return" is so far removed from the actual return values, that it takes some parsing (maybe you don't even see everything on an 80-column screen, or even worse, you split up one expression over several lines..

(Basically, I don't like multi-line expressions. Avoid stuff like

```
        x = ...

                + ... - ...;
```

unless it is _really_ simple. Similarly, some people split up their "for ()" or "while ()" statement things - which usually is just a sign of the loop baing badly designed in the first place. Multi-line expressions are sometimes unavoidable, but even then it's better to try to simplify them as much as possible. You can do it by many means

1. make an inline function that has a descriptive name. It's still complex, but now the complexity is _described_, and not mixed in with potentially other complex actions.

2. **Use logical grouping. This is sometimes required especially in "if()" statements with multiple parts (ie "if ((x || y) && !z)" can easily become long - but you might consider just the above inline function or #define thing).**
3. **Use multiple statements. I personally find it much more readable to have**

   ```
   if (PageTestandClearReferenced(page))
           goto dispose_continue;
   if (!page->buffers && page_count(page) > 1)
           goto dispose_continue;
   if (TryLockPage(page))
           goto dispose_continue;
   ```

   rather than the equivalent

   ```
   if (PageTestandClearReferenced(page) ||
           (!page->buffers && page_count(page) > 1) || TryLockPage(page))
                   goto dispose_continue;
   ```

   regardless of any grouping issues.

Basically, lines-of-code is a completely bogus metric for _anything_. Including maintainability.

3. Using Kernel Headers In Non-GPLed Code

28 Aug - 30 Aug (13 posts): If loadable modules are covered by Linux GPL?

In the course of discussion, Simon Richter [*] said he thought it would be OK to include kernel headers in a non-GPLed program. But Mike A. Harris [*] pointed out that some header files contained inline GPLed functions. Any program using those functions, he pointed, would then also have to be GPLed. Simon replied that this should be resolved as soon as possible, and Stuart Lynne opined that anything in the header files should be free for any use. At one point in the conversation, Richard B. Johnson [*] quoted Linus Torvalds [*] in the 'COPYING' file, where Linus gave his interpretation: **"NOTE! This copyright does *not* cover user programs that use kernel services by normal system calls - this is merely considered normal use of the kernel, and does *not* fall under the heading of "derived work". Also note that the GPL below is copyrighted by the Free Software Foundation, but the instance of code that it refers to (the Linux kernel) is copyrighted by me and others who actually wrote it."** Richard explained:

> **If I make a module, that cost my company hundreds of thousands of dollars to develop (back-projector for a CAT Scanner comes to mind), there is no way in hell that I will give this to a competitor by releasing the source-code. Even though a competitor will find it useless without the hardware to which it interfaces, the competitor will gain knowledge of my product(s), build them without investing in any Engineering costs, and put me out of business.**
>
> **Even M$ doesn't require that I give proprietary information away. If Linux wants to become the new standard for the computing industry, GPL or whatever can't claim any ownership of the work a company has done while using it.**
>
> **The Linux kernel didn't come with a "back-projector" module. Therefore, when I invest the time and money to make one, I certainly don't have to give it away.**
>
> **It is quite different if, for instance, I need to use a built-in system such as TCP/IP or SCSI and in the progress of my work I add functionality or fix bugs. Under GPL, I am required to supply any such**

changes (if asked).

This was not the final word, and several people pointed out the while Richard's point might be valid, it still did not change the status of GPLed code. There was no real resolution though...

4. Large RAID Under Linux

28 Aug - 4 Sep (14 posts): 2T for i386

Greg Hennessy asked (for his boss) if Linux could handle a 2 terabyte RAID partition, made from 32 disks of 72G each. He'd heard some conversations that seemed to indicate it should be possible, but he wanted to know if anyone had actually *done* anything like that on i386 yet. Alan Cox [*] replied:

> **At 2Tb in a single partition you might well start hitting barriers. I think there is a 1Tb limit per device somewhere. You also need to ask yourself how long 2Tb would take to fsck on a power failure. Right now 2.2 doesnt support journalling over software raid so that would stop you using reiserfs and ext3.**
>
> **You might be able to do that with hardware IDE raid controllers and the like such as the 3ware 8 port cards, or scsi raid controllers and then run ext3 or reiserfs.**
>
> **2Tb you will be pushing the envelope with 2.2.**

To help explain the 'fsck' time requirements for such a thing, Chris Good [*] gave his own experiences:

> **To give you a datapoint we have .5Tb arrays both hardware and software and they will both take around 10 hours to resync and a good 30mins to fsck, even mounting them will take 30 seconds or so. If you have lots of data on your discs then the fsck will take longer still. For large discs you *really* want journalling and you really don't want a power outage/disc failure to cause a resync.**
>
> **For filesystems of this size if you really care about**

> **downtime you might want to look at some of the commercial solutions eg Network Appliance NFS servers (http://www.netapp.com (http://www.netapp.com)). We've measured a netapp box with 1.4TB at 2 minutes from pulling the power to back online and serving data.**

But Ricky Beam [*] also replied to Alan's assumption that Greg would be using IDE. This seemed insane to him, and he also pointed out Alan's assumption of software RAID as opposed to hardware, and even the assumption of using ext2. But Alan replied, **"Im planning on deploying a 1Tb IDE raid using 3ware kit for an ftp site very soon. Its very cheap and its very fast. UDMA with one disk per channel and the controller doing some of the work."**

Joel Jaeggli [*] added that IDE looked especially attractive when considering size and cost requirements. He calculated, **"75GB scsi disks are 11 platter (ibm) or 12 (seagate) half height drives vs the 75gxp ide (ibm) with has five platters and is 1" high. The 75GB scsi disks are around $1500ea vs the ide which are around $550. so if you're requirements are lots of disk space you can do ide in about 1/2 the physical space and for about 1/3 the cost of a similar scsi implementation, at this time. there are of course still good reasons to go with scsi for various applications, but raw space alone probably isn't one of them."**

5. 2.4.0-test8-pre1

28 Aug - 29 Aug (4 posts): test8-pre1

Linus Torvalds [*] announced:

> **Ok, there's a test8-pre1 out there in testing. This is mainly for two reasons:**
>
> - **the mail-server I used got updated, and while all my mail was moved over to the new faster server, none of the mail status was saved, so I lost my "read vs new" mail information. And I'm too lazy to go back and try to see what I've read and what I haven't. So people who want a patch into the kernel should just check**

whether it made it, and re-send if not.

- talk is cheap. The thread group stuff is there. But it will be removed unless somebody actually takes a look at whether it really makes pthreads easier and more efficient.

  Most of the thread group code is trivial. The biggest impact was on signal.c, and even that wasn't because the new code is hard, it's just because I split up the "signal sending function from hell" into many smaller functions.

Whatever.

test8:

- pre1
  1. Oops. Moved back stallion.c to drivers/char. It's not a TV driver. Never has been, and I don't see it ever really becoming one ;)
  2. mca.c: outp_b() should be outb_p(). Obviously nobody actually _uses_ the MCA bus any more ;)
  3. umsdos should be ok again after the page_address() type-changes.
  4. re-enable asynchronous read-ahead code.
  5. Sun ESP driver update
  6. netfilter debug fixes
  7. IPv6 needs to register before proto_init()
  8. socket() error code fix (EAFNOSUPPORT instead of EINVAL)
  9. potential TCP socket leak fix
  10. don't self-deadlock on the kbd_controller_lock when probing for the mouse
  11. CONFIG_SMB_NLS_REMOTE didn't work. Silly typo.
  12. scheduler wakeup race condition could cause delayed scheduling on SMP..
  13. net/packet/af_packet.c: use the standard macros for marking page resevredness
  14. ncpfs buffer-overflow fix
  15. thread groups, take 1.

16. USB storage driver update

To item 9, David S. Miller [*] reported that the fix was not actually in the patch, and said he'd resend it later.

6. Alan Moving Ahead With 2.2.18 Before Releasing 2.2.17

31 Aug - 1 Sep (21 posts): Linux 2.2.18pre1

Alan Cox [*] put out 2.2.18pre1 and remarked, **"Since 2.2.17 isnt out yet I've released 2.2.18pre1 versus 2.2.17pre20. So you need to grab 2.2.16 then apply the 2.2.17pre20 patch then the 2.2.18pre patch of choice."** Remember in Issue #83, Section #9 (**24 Aug:** "Nearing 2.2.17") Alan was waiting for Linus' final approval. I suppose it never came...

Marco Colombo [*] asked, **"One day I hope you'll explain to us why this is not 2.2.17pre21... Either you are sure pre20 is going to be the official 2.2.17, or I'm missing something. And now that I look closer, you used the word 'Fix' a lot in the changelog. It won't be a great idea to release 2.2.17 with many outstanding bugs (however little they are...)."** Alan replied, **"The 2.2.18pre1 changes are higher risk problems to fix. 2.2.17pre20 is extremely solid. Its probably the most solid 2.2 kernel so far overall (Im sure it will break for someone though). The 2.2.18pre stuff is now starting again making higher risk changes that should be reliable but might produce suprises and need to be fixed/reverted before 2.2.18 final."**

7. Some General Discussion About Patch Submissions

1 Sep (6 posts): Small bugfix in ext2/namei.c

Jan Kara [*] sent an 'ext2' patch to Linus Torvalds [*]; and Dr. Michael Weller suggested that it would be better to send such things to Alan Cox [*] for review before sending it to Linus, because Michael had had some doubts about the patch's quality. Theodore Y. Ts'o [*] replied, **"I looked at it and was convinced. I wish Jan hadn't sent the patch directly to Linus, though, since I had a few other patches that I was going to send to Linus directly. Now depending on whether or not he accepted your patch, he may reject my patch entirely and I'll have to wait until test8-pre2 before I can**

figure out what to base my patch off of..... Ah well, life in the big city when the central maintainer refuses to use CVS....." And Alan added, **"For 2.2.x you can submit me stuff. Im not following 2.4test currently so 2.4 stuff that doesnt help. Nevertheless it should go to an obvious maintainer first. If I get 2.2 serial patches I'll still bounce them to Ted for example."**

8. 2.2.18pre2 Also Released Before 2.2.17

1 Sep - 2 Sep (6 posts): Linux 2.2.18pre2

Alan Cox [*] announced:

> **Since 2.2.17 isnt out yet I've released 2.2.18pre2 versus 2.2.17pre20. So you need to grab 2.2.16 then apply the 2.2.17pre20 patch then the 2.2.18pre patch of choice.**
>
> **Ok 2.2.18pre2 merges the large stuff that doesnt interfere with other parts of the kernel and some small fixes that are needed. Microcode.c also now compiles correctly. Pretty much all of this is stuff vendors are already shipping so while it is experimental (paticularly some of the USB) it shouldnt impact mainstream code. The AGP/DRM stuff is 2.4 compatible and works out of the box with the same Xserver setup on the i810 as on 2.4.**
>
> **2.2.18pre2 (versus 2.2.17pre20)**
>
> 1. **Fix the compile problems with microcode.c (Dave Jones, Daniel Rosen)**
> 2. **GDTH driver update (Achim Leubner)**
> 3. **Fix mathsemu miuse of casting with asm (??)**
> 4. **Make msnd_pinnacle driver build on Alpha**
> 5. **Acenic 0.45 fixes (Chip Salzenberg)**
> 6. **Compaq CISS driver (SA 5300) + cleanups + gcc 2.95 fixup (Charles White, me)**
> 7. **Modularise pm2fb and atyfb**
> 8. **Upgrade AMI Megaraid driver to 1.09 (AMI)**
> 9. **Add DEC HSG80 and COMPAQ 'logical volume' to scsi multilun list**

10. **SK PCI FDDI driver support (Schneider & Koch)**
11. **Linux 2.2 USB backport, backport 3 + further fixes from the USB list + mm/slab.c fix for cache destroy (Vojtech Pavlik)**
12. **AGP driver backport, DRM driver backport (XFree86, Precision Insight, XiG, HJ Lu and others)**

9. 2.0.39-pre7

2 Sep (3 posts): [Announcement] pre-patch-2.0.39-7

David Weinehall [*] announced:

> **This is the last 2.0.39 pre-patch, unless someone comes up with something horrible. Still, I'm not impossible to convince, should there be something one of you out the consider pressing. Give this one a tough beating, ok? Test and enjoy!**
>
> **2.0.39pre7**
>
> - **Fix a bug in af_unix that wrote to a socket after freeing it (aka the Win9x-related oops) (Michael Deutschmann)**
> - **Fixed typo in Documentation (Martin Douda)**

We Hope You Enjoy Kernel Traffic