ZACK'S KERNEL NEWS

Status of LinuxPPS

Udo van den Heuvel asked for the status of LinuxPPS (Linux Pulse Per Second): Why was it being rejected for inclusion in the kernel? Alan Cox and Andrew Morton scratched their heads and said they couldn't remember what if any objection anyone had had to the code. They both suggested resubmitting it because that would trigger any remaining alarm bells that had ceased to echo in the minds of anyone who cared.

The LinuxPPS (*http://wiki.enneenne.com/index.php/LinuxPPS_support*) API provides an interface between kernel and user space across character devices. A couple of weeks after that little exchange, Rudolfo Giometti submitted the core LinuxPPS code for inclusion. His idea was to make sure everyone signed off on the basic features; at least then there would be a big wad of code in the kernel for the PPS developers to add onto piecemeal.

At this point Andrew asked Rudolfo to explain which ancient objections, if any, remained unaddressed in the code. But Alan said he certainly liked this latest version. In response to Andrew, Rudolfo said he had fixed all objections, the sole objection being something from "George Spelvin" that all parties had agreed could wait until later. With no further debate on the issue, it seems likely that LinuxPPS – at least the core code – will soon be merged.

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this task is Zack Brown.

PMM

Michal Nazarewicz announced PMM (Physical Memory Management), code that allows users to allocate large contiguous regions of physical memory. For Michal at Samsung, this allowed him to decode and scale JPEG images and pass the scaled images to an X server, minimizing memory usage and increasing efficiency. Peter Zijlstra remarked that this might be all well and good, but if no contiguous regions of memory were available for allocation, Michal's PMM code would fall down flat. After a short bit of uptime, few systems would have any large contiguous areas of RAM to choose from. Michal replied that PMM reserved a large pool of RAM at boot time and allocated memory from the pool as needed. Peter asked how Michal would stop the PMM reserved RAM area from fragmenting as it is used. Michal replied that different use cases would result in fragmentation and the only sure way to avoid this would be to increase the size of the RAM pool reserved at boot time. But that was not so scalable. PMM, Michal said, was an attempt to find a compromise between the various needs of the running system. But, Andrew Morton remarked, "We do have capability in page reclaim to deliberately free up physically contiguous pages (known as "lumpy reclaim"). It would be interesting were someone to have a go at making that available to userspace." Michal grabbed hold of that idea and started a technical discussion about its feasibility. It does seem that in its current form, PMM would not be acceptable in the kernel because of the very restricted and specialized use case it represents.

In-Kernel Debugger Status

Jason Wessel proposed unifying KDB and KGDB, essentially making KDB a front end to KGDB. He tried to put the idea as delicately as he could, asking whether the KDB folks would still find value in such a project. He posted some patches along the lines of what he had in mind. Maxim Levitsky and Louis Rilling both jumped up to say that they liked KDB and would definitely love to see it in the kernel. Christoph Hellwig also expressed enthusiasm for the idea, adding that making KDB a front end to KGDB would be fine with him. Martin Hicks was also excited about this prospect. In this thread at least, the consensus seemed to be that having a native kernel debugger would be excellent and merging KDB and KGDB in the way Jason suggested would also be excellent. On the other hand, Linus Torvalds has resisted including a native debugger in the kernel, and he certainly won't want to let a new front end in until his own objections are addressed. A week or so later, Jason posted more patches, and Ingo Molnár offered some fairly invasive criticisms, remarking, "I supported and helped a debugging back end and I don't consider a front end completely impossible either. But it will have to meet a lot of stringent standards because a good kernel debugging front end's cross section to the system is even larger than a back end's. It's a tough job to get this done." Jason responded with an attempt to address some of Ingo's objections, but a big effort will have to be put into this before it will make it into the kernel.

Driver Coding Pitfalls

Atul Mukker from LSI Corp. announced their intention to initiate a new approach to LSI RAID controller driver creation. They want to keep the code generic across multiple operating systems, keeping only small Linux-specific, Mac-specific, and so on, sets of surface code to access the core driver. He asked for any advice the Linux kernel community wanted to give him.

Jeff Garzik agreed that this could be a great benefit to everybody, if done right. But he did feel that certain mistakes had been made in the past, that they would be best not to repeat. He said, "in the past, when hardware vendors have created a cross-OS layer for their drivers, that layer wound up decreasing performance, increasing code size, introducing bugs, and decreasing overall portability."

He pointed to Intel's network drivers as an example of a similar effort that had avoided the worst pitfalls. Atul was heartened by this reply because it seemed that it was indeed possible to do what he'd envisioned. He asked for further suggestions, and Jeff obliged. Jeff suggested making the code modular, to keep code supporting specific hardware separate from each other and separate from the OS-specific code. This might not be a simple thing to do, Jeff said; in fact, he considered it a first-class engineering task. As a general admonition, he recommended avoiding too many C pre-processor wrappers and recommended making good use of C's native types and enums.

Jeff also recommended that any code that could be generalized and made non-specific to LSI's drivers should be kept out of the driver, so it could more easily be shared by other projects. Jeff said that the driver's ABI (Application Binary Interface) should be consistent with other Linux drivers. Features not unique to LSI but similar to features found in other drivers should try to behave the way those other drivers behaved. Features unique to LSI, Jeff said, were fair game for LSI to handle however it wanted. Jeff concluded that Linux contributors had to consider LSI's code within the context of code submitted from other hardware vendors, including LSI's direct competitors. This was primarily to avoid code duplication, but also because multiple implementations of the same features would multiply the potential bugs. The Linux maintainers also had to consider the future case in which hardware would no longer have vendor support, but users would still rely on the Linux drivers. Atul thanked everyone for helping them get started with this project.

Event Configuration In DebugFS

Steven Rostedt announced enhancements to event tracing in DebugFS. Enabling events one at a time can be tedious, and even enabling them in groups requires too much detail. Steven wanted to be able to enable all tracing events below a given directory in DebugFS, so his patch created a new file in each directory called *enable*. A value of *1* in such a file will enable all events defined in that directory and all subdirectories. A value of *0* will disable the same set of events. But he didn't stop there! To prune the directory tree selectively, you can put a *1* in the *enable* file of a given directory and then a *0* in that of any subdirectory to be excluded. So, with a minimum of effort, swaths of events can be enabled and excluded, without the bother of naming them all. Frederic Weisbecker loved this patch, but Li Zefan objected, saying Steven's implementation made it difficult to figure out which events were enabled. He said that he'd normally expect to get a list of enabled events by viewing the config file, but in Steven's patch, the config file would mysteriously contain only a *1* or *0*. Steven's response was that the patch had taken him 15 minutes to code, claiming the benefit of simplicity, and that it was tailored for the person setting the events rather than anyone interested in reading them. Because the configuration files referred to a hierarchically organized directory tree, all the information was still available and could be retrieved by a script. Li replied that he had no serious objection to this, but had brought it up in case anyone else wanted to take issue with it. Apparently, no one did.