Zack's Kernel News

Chronicler Zack Brown reports on the latest news, views, dilemmas, and developments within the Linux kernel community.

By Zack Brown

KVM API Revamp

Avi Kivity felt that the KVM (kernel-based virtual machine) programmer interface was getting gradually out of control, and needed to be reined in. He really wanted to ditch the old API entirely and start fresh with a new, well-thought-out interface that did not have the years of cruft and creep clinging to it that the current KVM did. But, because he felt that a total revamp might be too wrenching for users, he just posted some of his thoughts about where such a thing might lead, if it were ever done.

First of all, he wanted to migrate away from using ioctls as the primary interface and switch to system calls instead. This would give a slight speed increase and would clarify the semantics by tying virtual machines to threads rather than file descriptors. Getting rid of ioctls is also considered generally good because they have utterly insane semantics and are generally considered to be an unmanageable, labyrinthine nightmare. On the other hand, introducing new system calls is also generally frowned upon.

The next item on Avi's hit list was the way the virtualized CPU state was handled. The current system used a mishmash of ioctls to get and set state in a highly disorganized manner. He wanted to replace all that with just two system calls, with a clear and predictable data structure.

He also wanted to eliminate a whole swath of the kernel-based hardware emulation that could be done just as easily in userspace. As Linus Torvalds has said: Anything that *can* be done in userspace, *should* be done in userspace. In particular, the IOAPIC (I/O Advanced Programmable Interrupt Controller) hardware and related hardware used for interrupt handling could be shunted out to userspace.

Memory management was another item on Avi's list. The current code managed all memory slots individually; in Avi's proposal, a single interface would replace the entire memory map all at once.

Avi's proposal stirred up a little controversy. A surprising number of people wanted to keep the ioctl interface, even though it's usually considered ugly and impenetrable. As Alexander Graf put it, "I really do like the ioctl model btw. It's easily extensible and easy to understand. I can also promise you that I have no idea what other extensions we will need in the next few years. The non-x86 targets are just really very moving. So having an interface that allows for easy extension is a must-have." In a later post, he added, "I'm trying to mostly add "generalized" ioctls whenever I see that something can be handled generically, like ONE_REG or ENABLE_CAP. If we keep moving that direction, we are extensible with a reasonably stable ABI. Even without syscalls."

Avi saw the wisdom of preserving extensibility, which he agreed would mean they'd have to wait at least until the interface thoroughly stabilized before considering a migration to system calls. And with KVM's non-x86 interfaces still growing rapidly, there'd be no way to know how far in the future that might be.

So, although a lot of people seemed to be generally in favor of simplifying the KVM interface, in practice, there are still reasons why it is the way it is that are tough to refute. And, migrating to something new might be more difficult and time-consuming than just deciding to have two separate interfaces side-by-side until the old one can be retired.

Load Balancing Improvements

Rakib Mullick has developed a new technique for load balancing between multiple CPUs on a given system. He calls it the Barbershop Load Distribution algorithm, or BLD. In the general case of just starting up new processes, the BLD resembles Linux's existing algorithm: whichever CPU has the least load gets the new process.

Where the BLD seeks to improve on the existing approach is in rebalancing already-running processes and deciding where to put processes swapping back into memory from disk. Rakib's idea is that the BLD will notice when the system load changes and, at that time, will reorder the linked-list of CPUs based on each individual load. This operation is apparently extremely inexpensive in terms of time; so, whenever a process needs to be migrated, it's an easy matter just to look up the least loaded CPU on the list. In algorithm-speak, this will take O(1) time, which means it will take the same amount of time regardless of how many CPUs are on a given system.

Essentially, as Rakib puts it, the BLD achieves load balancing without doing any ac-

ZACK BROWN

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**.

tual load balancing because it only moves process around when they are first created or when they're woken up. If those things don't happen, the BLD assumes that the load has already been properly assigned.

No one had any serious objection to his code – although that's not to say it'll be going directly into the official tree. Hillf Danton had a bunch of minor nitpicks with the code, and Rakib is still making improvements and doing lots of cleanup to his own implementation. But, at some point, it will probably be given serious consideration as a replacement for the current load balancer.

Scheduler Testability

Other folks have also been working on scheduler stuff lately. Pantelis Antoniou has been a little frustrated by the difficulty of making any kind of scheduler improvements. His particular area of interest is with the big.LITTLE devices that have been coming out over the past year. The idea behind these devices is to have a very powerful CPU sit side-by-side with a much less powerful CPU and let the OS seamlessly migrate processes between them. This way the bigger CPU could be powered down a lot of the time, thus extending battery life for phones and pads and other modern devices.

The problem is that the big.LITTLE idea is very different from standard systems, so it's hard to make sure that scheduler improvements under that kind of architecture would also improve, or at least not ruin, scheduler behavior on the more common systems. The problem goes deeper than that, though. It's not really a question of strange new hardware configurations. The scheduler can have vastly different behaviors depending on the user's normal behavior. Each user uses a system differently, favoring different software and displaying different habits of switching between activities. Optimizing the scheduler for just one of those use-cases would not in any way guarantee an improvement for other use-cases.

The scheduler is therefore a pit of despair, where any change could be catastrophic, and it's very difficult to find meaningful ways to test a given change. At the same time, however, everyone wants to see massive improvements in the scheduler, to reduce choppiness on loaded systems, and to enable real-time applications like complex medical devices that control life-and-death situations. And, for video playback.

Pantelis thought that one way to address the problem would be to develop a system of recording a particular session of computer usage, such that it could be replayed on a different computer. If, for example, a million people contributed their common working behaviors to a centralized repository, new scheduler code could potentially be tested by replaying those use-cases with the new scheduler and measuring statistical improvements.

Frederic Weisbecker pointed out that the `perf sched` tool was able to do rudimentary recording and replaying of a given use session. Pantelis said that this would probably be a good starting point for the project. But, they both agreed that the `perf sched` tool would need a lot of enhancement before it would really be ready for what Pantelis had proposed. And, as Stephan Bärwolf pointed out, any sort of recording feature would dramatically slow down the system using it, which might have an effect on user behavior, thus tainting the value of the recording.

In spite of all the difficulties, Pantelis's idea does represent at least a clear way to provide scheduler developers with some sort of mechanism to test their work on a larger scale than is currently possible. ■■■