Zack's Kernel News

Numbering devices

One of the big controversies in Linux kernel development is the device numbering system. Hardware manufacturers keep pumping out new devices, and computers keep coming out that can hold more and more of them, and the operating system has to keep track of them all somehow.

So far, this data has been defined in the source tree and stored in some small variables; but for a long time there has been a movement to ditch the static numbering system, in favor of one that defines these numbers dynamically for each running system.

This is not as easy as it appears, as demonstrated by the fact that, while Linus Torvalds strongly favors that approach, many other developers including Alan Cox have chosen to disregard that preference in the 2.4 tree, and to continue extending the static numbering features to accommodate the rising availability of devices and scalability of individual systems. In April, Andries Brouwer proposed a patch to increase the size of the kdev_t variable, used to hold these device numbers, from 16 bits to 64.

It's generally agreed, even among critics of static numbering, that 16 bits is starting to get uncomfortably tight, while 64 bits would satisfy all needs for many years to come. There are technical problems with any solution, including ones that favor static numbering, although they are not as serious as those confronting dynamic numbering. But dynamic numbers have the advantage of being a permanent solution, while any static numbering system will one day run up against the same problems that confront it now.

Even critics of dynamic numbering agree that it's not a bad idea, just very difficult to get right. Folks like Roman Zippel have been arguing very strongly that the kernel is rapidly approaching a time when dynamic device numbering will be a real possibility; and that some parts of the kernel, like the SCSI code, already successfully use dynamic numbering.

He argues that as long as the 16 bit device numbering scheme will hold out, developers should wait and push for the 'proper' solution.

But other developers say they need a solution in the very short term, to deal with real life hardware that needs to work properly now, not in a few years after the next development cycle is completed. For now, the controversy continues unresolved. ■

Compressing code

Data compression is often proposed whenever space gets tight. Most Linux kernel binaries are compressed on disk, and uncompressed at boot time; but there are many opportunities to eek out a few more bytes here and there. The question is always, is the amount of space saved worth the additional time it takes to compress and uncompress it?

Two attempts were made in April to add compression features to the kernel. The first did not turn out so well; the second still has a chance. Timothy Miller had the idea that, instead of swapping to disk, data could be compressed in RAM, thus freeing up space for other processes without incurring the overhead of disk accesses. Because a hard disk contains a rotating slab of metal that must be read from and written to by moving a large sliding arm, where as RAM has no moving parts, it should be clear that avoiding disk accesses has the potential to really speed things up in just about any situation. It's not unreasonable to think that data compression might be faster than disk accesses; in fact, the original Mac OS implemented such a thing and called it 'RAM Doubler'. Inaky Perez-Gonzalez had also made his own attempt under Linux in the 2.2 days, and though he'd successfully shrunk RAM usage, the machine slowed to a crawl.

A more recent attempt, hosted by the Department of Computer Science at the University of Sao Paulo, in Brazil, has also had success, although the project seemed to lose momentum in late 2002. It also suffered from technical limitations, e.g. not supporting SMP or pre-emption. Timothy actually tried the code. For systems with large amounts of RAM, it seemed to have a negative impact on overall speed.

The second attempt at data compression in April took a different tack, though it came from the same person. This time, instead of targeting all of memory management, his idea was to compress kernel text messages, by identifying the most common words and replacing them with single characters, which would then be substituted back to the original words when needed. It turned out that this kind of string compression was very popular in the old days, when every byte counted; so a variety of real, working methods are available for such things.

After a few first attempts, Timothy found that he could save as much as 80K in the compiled binary. Although there are still questions about how to integrate this into the build system, it seems clear there is something to be gained by it. ■

INFO

The Kernel Mailing List comprises the core of Linux development activities. Traffic volumes are immense 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 that take on this impossible task is Zack Brown.

Our regular monthly column keeps you up to date on the latest discussions and decisions, selected and summarized by Zack. Zack has been publishing a weekly digest, the Kernel Traffic Mailing List for several years now, reading just the digest is a time consuming task.

Linux Magazine now provides you with the quintessence of Linux Kernel activities straight from the horse's mouth.

USB Gadget

In March, David Brownell announced a new kernel API, to be used by USB peripherals that embed Linux within themselves. Drivers using this "USB Gadget" API will be portable to a wide variety of devices, while USB hardware that embeds Linux will be able to make use of a wide variety of device drivers.

This project comes out of the strange situation, that many USB devices can themselves run Linux, and so ambiguities creep into the discussion, when talking about Linux 'support' for a given device. Does it mean that a Linux system can communicate with that device, or that the device itself can use Linux to communicate with other systems?

To clear up the issue, many USB developers have agreed to call drivers running on the main host system, "USB device drivers"; while drivers running on the peripheral devices themselves are called, "USB gadget drivers". David's API is for use by "gadget drivers".

Most embedded systems have many unique qualities, born out of the marketing and technical requirements for the gadget. As such, it can be tempting to write drivers specifically for the particular hardware involved. The drawback of this is that this software may be very difficult to port to other embedded systems.

David's new API provides a generic set of function calls that can be used by any gadget driver to communicate with the underlying hardware. As such, developers coding for one gadget can very easily migrate their work over to a new gadget without having to start over from scratch.

User-Mode Linux

Kernel debugging has become more and more difficult over the years. With support for more and more processors, and greater and greater threading capabilities, the problem of identifying exactly where a problem occurs has become vastly more complex.

Couple this with the fact that, while a user program can crash and be restarted fairly quickly, a kernel crash generally means several minutes wasted during a reboot. One way around this has been to run the kernel on itself as a user process.

User-Mode Linux, or UML, maintained by Jeff Dike, has gradually become a significant debugging tool, because of its ability to boot and control kernel images very quickly. Recently Werner Almesberger produced a new project, called UMLSIM, which extends UML to allow fine-grained control over the flow of time as it appears to the running kernel.

Using UMLSIM, it is possible to set breakpoints within the kernel itself, and control its behavior from outside. This can be used to call functions, force functions to return, and read and write variables and arguments. As it turns out, Karim Yaghmour and others have been working along similar lines, producing a project they call genevent.

While this improves upon Werner's breakpoint feature, UMLSIM still goes a bit farther than genevent, in terms of minimizing the number and kind of changes that must be made to the kernel, in order to make use of these features. Perhaps UMLSIM and genevent will feed off of each other, or even merge into one project at some point.

Radeon fork

Benjamin Herrenschmidt has forked the Radeon Framebuffer code base away from Ani Joshi. Ani apparently stopped applying patches and virtually disappeared sometime in mid 2002, which became frustrating for some folks. Finally after about a year, Ben announced the fork and put out his own release, incorporating a number of patches.

In his announcement, he said he would continue to maintain the fork until Ani either officially stepped down as maintainer, or else incorporated all the patches that had been sitting around for so long. Before the fork, the Radeon code had been somewhat usable, though still clearly in need of development. Problems included incorrect display of the cursor, crashes, and other issues.

This is par for the course of any development effort, and under Ben the same technical problems remain, though folks have already noticed improvement in various areas. Ben's plan is apparently to continue to maintain the existing codebase in the 2.4 kernel series, and do a complete rewrite for 2.5; the 2.5 tree, at least in theory, is in a feature freeze in preparation for 2.6 or 3.0; according to tradition, that may or may not mean anything.

It is possible that a Radeon rewrite could still make it in before the next stable series. As of this writing, Ani has yet to be heard from on the issue. Typically, code forks have been seen as justified when the current maintainer fails to accept good patches or put out timely releases; while developers make every effort to provide those patches to the maintainer in a useful way. This seems to be the case here, and as a result no one seems to have any complaints about Ben's decision.

Unfortunately, not all code forks can be so reasonable or peaceful. Albert D. Cahalan's and Rik van Riel's diverging procps trees can attest to that.

Theirs was also a case of the maintainer just dropping out, although in the procps situation, the code languished for quite, and was finally picked up quietly by multiple people, who discovered the redundancy only later, and then were unwilling to merge their work.

More disks

Recently, Badari Pulavarty offered up a patch to overcome the limitation on the number of disks that could be simultaneously connected to a Linux system. He beat the value of 256, leaving no known value in its place. In his tests, he managed to do simultaneous I/O on 4000 disks before using up all of his memory.

So as far as anyone knows, the maximum number of disks is now limited only by the resources available to support it. Each disk must be represented in the system by a data structure, which uses up RAM; and these data structures must be traversed, in order to give the user access to them; which uses up time.

For journaled filesystems there is the problem of dealing with the journal data.

Badari had given everyone a 4000 disk mountain, and they were going to climb it "because it was there." This situation also goes back to the 'static vs. dynamic device numbers' controversy, since it is now becoming practical to have an ungodly number of devices attached to a given system at any time.