Debugfs and the making of a stable ABI

By **Jonathan Corbet**
December 3, 2008

Remi Colinet recently proposed the addition of a new virtual file, `/proc/mempool`, which would display the usage of memory pools within the kernel. Nobody really disagreed with the idea of making this information available, but there were some grumbles about putting it into `/proc`. Once upon a time, just about anything could go into that directory, but, in recent years, there has been a real attempt to confine `/proc` to its original intent: providing information about processes. `/proc/mempool` is not about processes, so it was considered procfile-non-grata. It was suggested that another home should be found for this file.

Where that other home should be is not obvious, though. Somewhere like `/sys/kernel` might seem to make sense, but sysfs has rules of its own. In particular, the one-value-per-file rule makes it hard to create an easy file where developers can simply query the state of a kernel subsystem, so sysfs is not a suitable home for this file either.

The next option is debugfs, which was created in December, 2004. Debugfs is meant to be an aid for kernel developers; it explicitly disclaims any rules on the types of files that can be put there. All rules except for one: debugfs is not a mandatory part of any kernel installation, and nothing found therein should be considered to be a part of the stable user-space ABI. It is, instead, a dumping ground where kernel developers can quickly export information which is useful to them.

Since debugfs is not a part of the user-space ABI, it seems like a poor place to put things that users might depend on. When this was pointed out, it became clear that the non-ABI status of debugfs is not as well established as one might think. Quoting Matt Mackall:

> The problem with debugfs is that it claims to not be an ABI but it is lying. Distributions ship tools that depend on portions of debugfs. And they also ship debugfs in their kernel. So it is effectively the same as /proc, except with the 1.0-era everything-goes attitude rather than the 2.6-era we-should-really-think-about-this one.
>
> Pushing stuff from procfs to debugfs is thus just setting us up for pain down the road. Don't do it. In five years, we'll discover we can't turn debugfs off or even clean it up because too much relies on it.

As an example, Matt pointed out the extensively-documented usbmon interface which provides a great deal of information about what's happening on a USB bus. If it is not an ABI, he says, nobody should be upset if he submits a patch which breaks it.

That is a perennial problem with interfaces between the kernel and user space; changing them causes pain for users. That is why incompatible changes to user-space interfaces are almost never allowed; an important goal for the kernel development process is to avoid breaking user-space programs. One might think that this problem could be avoided for a specific interface by explicitly documenting it as an unstable interface. The files in `Documentation/ABI/testing` are meant to serve that role; anything found there should be considered to be unstable. But, as soon as people start using programs which depend on a specific interface, it has, for all practical purposes, hardened into part of the kernel ABI.

Linus put it this way:

> The fact that something is documented (whether correctly or not) has absolutely _zero_ impact on anything at all. What makes something an ABI is that it's useful and available. The only way something isn't an ABI is by _explicitly_ making sure that it's not available even by mistake in a stable form for binary use.
>
> Example: kernel internal data structures and function calls. We make sure that you simply _cannot_ make a binary that works across kernel versions. That is the only way for an ABI to not form.

So a given kernel interface can be kept away from ABI status if it is so hard to get to, and so unstable, that nothing ever comes to depend on it. The kernel module interface certainly fits this bill. Modules must generally be built for the exact kernel they are intended to work with, and they must often be built with the same configuration options and the same compiler. Anybody who has gotten into the dark business of distributing binary-only modules has learned what a challenge it can be.

Debugfs is different, though. It is enabled in a number of distributor kernels, even if, perhaps, it is not mounted by default. Once a set of files gets placed there, their format tends to change rarely. So it is possible for people to write programs which depend on debugfs files. And the end result of that is that debugfs files can become part of the stable kernel ABI. That is generally not a result that was intended by anybody involved, but it happens anyway. The only way to avoid it would be to deliberately shake up debugfs every kernel cycle - and few developers have much desire to do that.

This is a discussion without a whole lot in the way of useful conclusions; it leaves `/proc/mempool` without a home. ABI design, it turns out, is still hard. In the longer term, dealing with an ABI which was never really designed, but which just sort of settled into being, is even harder. There does not appear to be any substitute for thinking seriously about every interface between kernel and user space, even if it's just for a developer's debugging tool.

Debugfs and the making of a stable ABI
Posted Dec 4, 2008 7:08 UTC (Thu) by **ntl** (subscriber, #40518) [Link]

I'll grant that "one value per file" (sysfs) and "no new process-unrelated files" (proc) are positions strongly held by a vocal minority of kernel developers, but, in practice, they don't appear to be rules that are well-enforced. I feel bad for the submitters of patches such as this one; one gets the impression of a double standard.

We know how to manage mutation, though...
Posted Dec 4, 2008 14:47 UTC (Thu) by **davecb** (subscriber, #1574) [Link]

Multics addressed that, back before Unix existed, as it needed to support controlled change.

Paul Stachour wrote about the process in Observations about Software Maintenance at multicians.org.

I've used it for avoiding "flag days" in new software whose API was still mutating, which sounds a lot like this case

--dave (who typed Paul's talk in) c-b

Debugfs and the making of a stable ABI
Posted Dec 4, 2008 16:57 UTC (Thu) by **felixfix** (subscriber, #242) [Link]

One thing I started long ago, but still don't always do, is add a version table to every database and make sure the first line of every flat file, or the name itself, includes a version. When I don't do it, usually because I mistakenly think the data is so short lived that it won't matter, I use the absence of version info to mean version 0 :-)

Needing two catch all file systems
Posted Dec 5, 2008 1:51 UTC (Fri) by **nevets** (subscriber, #11875) [Link]

Ftrace has found its home currently in debugfs. And the main reason is that we expect to be able to change the output when we feel fit. Ftrace is still new, and we are still learning what the best ABI should be. Until we have a stable ABI, I do not want user space tools to rely on it yet.

What would be nice is to have another filesystem like debugfs. Maybe a "stablefs" that things can move from debugfs to stablefs when the ABI has been set in stone. That would be a nice way to let user application developers know what they can rely on.

Maybe rename debugfs to unstablefs.

Debugfs and the making of a stable ABI
Posted Dec 6, 2008 20:21 UTC (Sat) by **giraffedata** (subscriber, #1954) [Link]

There's something deeply disturbing about a practice of ensuring forward compatibility by withholding function from the present release. It's selfishness masquerading as public service.

It *is* a software distributor's duty not to *mislead* people into using a function that will make it harder to move to a later release, and that's where the documentation that Linus finds useless comes in.

I think the only reason Linus sees documentation as having no effect on what people depend upon is that the Linux kernel has a no documentation culture. Stating in Documentation/xxx.txt that X isn't intended as a stable API falls way short of informing users that X isn't intended as a stable API, and in fact there often is no way to get that information across. In fact, given the vague nature of the Linux development community, I'm not sure "intended" has much of a meaning. But I'm sure Linus knows of many features in other software that are not depended upon only because documentation warns of loss of forward compatibility.

Debugfs and the making of a stable ABI
Posted Dec 7, 2008 18:03 UTC (Sun) by **bronson** (subscriber, #4806) [Link]

Linus's point is that when he documents a feature "unstable" or "do not use", distros often use it anyway. Once that happens, it doesn't matter how it's documented, the interface must be preserved forever.

It's not "Linus finds warnings in documentation useless" as you suggest, it's "Linux users have demonstrated time and time again that it's useless". A subtle but important difference.

The only practical solution so far has been to simply withhold the feature until the interface is stable. But all programmers do this. I don't see why you find that so deeply disturbing...?

Debugfs and the making of a stable ABI
Posted Dec 7, 2008 23:36 UTC (Sun) by **giraffedata** (subscriber, #1954) [Link]

> Linus's point is that when he documents a feature "unstable" or "do not use", distros often use it anyway.

He makes a more general point. He says whether something is documented has zero impact on anything. While I believe from context he meant to limit that to impact on whether it's OK to change function in a later release, I believe he meant to cover engineering in general and not just the Linux kernel, and that he meant zero.

Even in the Linux kernel, there are plenty of cases where because something is documented as something developers want to change in the future, a Linux user did not use it. But if you consider things that don't have Linux's no-documentation culture, there are even more cases of someone not using an interface because documentation shows it is not intended to be an interface.

Linus' statement is, like so many he makes, greatly exaggerated.

> The only practical solution so far has been to simply withhold the feature until the interface is stable. But all programmers do this. I don't see why you find that so deeply disturbing...?

Neither of those things affect whether there's something disturbing in the practice. What I find disturbing is the contradiction or irony. Distributing tools is primarily about enabling people. Tool makers shouldn't be in the business of manipulating behavior.

It reminds me of an I/T group I once worked for, whose mission was essentially to withhold computing services from people (it was a large centralized I/T department that functioned as a budget control point).

Debugfs and the making of a stable ABI
Posted Feb 25, 2011 5:07 UTC (Fri) by **jrn** (subscriber, #64214) [Link]

> What I find disturbing is the contradiction or irony. Distributing tools is primarily about enabling people. Tool makers shouldn't be in the business of manipulating behavior.

It _is_ interesting. But I think you'll find something similar in the behavior of many tool makers (toy makers, carpenters, machinists, etc). It is considered irresponsible to give people something that will only lead to pain or danger.