Submitting Patches for the Mainline Linux kernel from CELF activities

This article describes some patches that were contributed by the CE Linux Forum to the mainline Linux kernel. The process used to submit these patches, and the results from their contribution is also described. Through these examples, several important points about submitting patches to open source projects are illustrated. It is hoped that by sharing this information, you will feel more comfortable sending in your own improvements to the Linux kernel and other open source projects.

Printk-times patch

The "printk-times" patch was developed and maintained by the Bootup Time working group of the CE Linux forum. This patch added a feature to the Linux kernel to include timing information on each line of kernel printout. This feature is valuable for determining the areas where the kernel is spending significant time during bootup.

The patch submitted by the working group was based upon a patch from a CELF member that had been written for a different version of the Linux kernel a few years previous, but which had never been submitted to mainline. To prepare to submit this feature to mainline, the first thing we did was to update the feature to apply to the latest version of the Linux kernel. Also, the feature was modified to use a new timer mechanism in the Linux 2.6 kernel that was architecture-neutral. That is, we tried to make the patch as generally useful as possible. Before submitting the patch, the feature was tested on Intel and PowerPC platforms. After this, the patch was formatted according to kernel developer preferences (see the Resources box), and sent to the Linux kernel mailing list.

The first person who accepted the patch and applied it to their tree was Andrew Morton. Andrew maintains a tree of "pending" patches that are being validated for final inclusion in the master source tree of the kernel. The master source tree, or "mainline" tree, is maintained by Linux Torvalds. After a patch has been in Andrew's tree for some period of time, if the patch seems acceptable, Andrew sends it to Linus for inclusion in the mainline tree. The printk-times patch was accepted by Andrew and placed in his tree with little discussion. One of the reasons for this was that the feature was quite isolated. The patch only touched 3 files, and the modifications it made were small. At this point, the CELF working group continued with other activities, but we watched the kernel mailing list for feedback on this patch.

A few weeks later, two different developers (not forum members) found some bugs in the code and reported them. These bugs were immediately tested and confirmed by a member of the CELF working group. One of the bugs resulted from a set of kernel messages that output only the message header (specifying the log level for the message), but no actual message

Resources for Submitting Patches

Here is a list of resources for submitting code to open source projects:

The CE Linux Forum maintains a page of tips and instructions at: http://tree.celinuxforum.org/CelfPubWiki/PatchSubmissionHowto

The kernel source tree contains guidelines for coding style and patch submission. In a Linux kernel source tree, see the files `Documentation/CodingStyle` and `Documentation/SubmittingPatches`.

Andrew Morton, an important kernel developer who manages thousands of patches for the kernel development process, wrote a set of guidelines for submitting patches. Andrew's "perfect patch" guidelines are at: http://www.zip.com.au/~akpm/linux/patches/stuff/tpp.txt

Another important kernel developer, Jeff Garzik, has also written a document with patch format guidelines. His document is available at: http://linux.yyz.us/patch-format.html

content. These messages were only output in rare instances by the SCSI driver of the kernel. Fixes were made to the patch and testing was done again. Within a day a new patch was sent to the kernel mailing list. The printk-times feature appeared in the mainline tree a few days later.

Printk-times lessons

This experience illustrates a few important lessons for submitting patches to the Linux kernel. First, it is important to test your code as extensively as possible before sending it in. Patches should be based on the latest version of the Linux kernel. This really is a pre-requisite before a patch will receive any attention from other kernel developers. In the case of the printk-times patch, one important benefit we received by submitting it was that independent developers tested it with their configurations. They would not have done this if the patch were based on an older version of the kernel. Since the patch was in a current test source tree (Andrew Morton's tree), it was very easy for them to turn on the option for the printk-times feature and notice a problem.

Another key point in the success of this patch was that it was formatted correctly, and it was small and isolated. Very often, patches will take a long time to be reviewed or discussed on the kernel mailing list. But this feature was accepted relatively quickly since it was simple and clean. The Bootup Time working group of the CELF had other related technologies we could have included in a single big “kernel instrumentation” patch, but because we submitted this patch separately, kernel developers were able to quickly review the code and it did not get “stuck”. Sometimes a patch gets stuck because it is large and difficult to understand, and other times it gets stuck because the developer does not take the time to format the patch correctly. The main kernel developers use many automated tools to manage the very large number of patches they receive. If their tools cannot process a patch, it is often dropped. Also, if a patch contains trivial formatting or style errors, this creates an unnecessary extra review cycle which ruins the momemtum of the submission.

Note that even though the printk-times patch was accepted into the experimental tree quickly, it took a few weeks before other developers reported problems and a little more time before it was finally accepted into the mainline tree. (This is actually quite fast. Many patches takes months and multiple rounds of submission and review to be accepted into mainline.) A key lesson here is that you need to be patient, and let other developers work at their own pace. However, once developers respond to your patch, you should act as quickly as possible to respond back to them. The open source methodology works in a highly distributed, highly time-disjoint fashion. E-mail is the primary means of communication among developers, and this creates an environment where discussions can be spread out over many days. This is actually a benefit for people for whom English is a second language, because they can process messages at their own pace.

Also, however, this creates an environment where a patch might not be noticed. Because of the high volume of traffic on some lists, many developers do not read every message, and they may not notice your feature immediately. Unfortunately, some patches from newcomers are not noticed or commented on by anyone. There is no need to worry. After a few weeks, the patch should be re-submitted again. Sometimes patches from even the most experienced kernel developers must be submitted multiple times before they are responded to.

In the case of printk-times, by submitting the patch, a few important bugs were found and fixed. Although the forum working group had done testing of this code, the particular bugs that remained were exposed by unexpected cases. These cases would not have shown up in continued testing, since embedded systems that use SCSI are rare. However, these bugs could easily have shown up later in more relevant kernel sub-systems, and caused problems that were difficult to find. Finding bugs, and getting feedback on extensions and improvements are some of the most important reasons for contributing work to open source projects.

Preset-LPJ patch

Another contribution which demonstrates these principles and others is the contribution of the “preset-LPJ” patch. This patch disabled the calculation of the “loops_per_jiffy” variable by the kernel. This operation takes a few hundred milliseconds in the normal case, and is one of the longer delays in kernel bootup. The patch eliminated the dynamic calculation of the value, and used a preset value instead (which is more appropriate for an embedded configuration of Linux, where the hardware is fixed and known.)

This patch had been maintained by the Bootup Time working group of CELF for many months. It was a very small patch, and it didn't seem important enough to contribute. The area of code that it modified had not been changed for many years, so there didn't appear to be much burden in maintaining the patch outside the main kernel tree. Eventually, however, this patch was formatted correctly and sent to the Linux kernel mailing list. Within hours, an independent developer noticed the patch and started testing it and discussing it on the list. A few different developers discussed different issues with the patch, including whether the preset value should be a compile-time option or a kernel runtime argument. Also, they discussed how to improve the messages for the system, to avoid confusion and help users select the appropriate preset value. Eventually, the independent developer re-submitted a modified patch, which Andrew Morton accepted into his tree. A few days later, this feature was included in the mainline tree.

A few weeks after all this, the code that the patch modified was refactored and moved into a different file in the mainline kernel source tree. Because the patch was already integrated, there was no work required by forum members to keep this patch up-to-date.

Preset-LPJ Lessons

One of the most important lessons of this experience is that other developers may provide great feedback and improvements to your features. In the case of the preset-LPJ feature, the feature was so simple that it didn't seem like it could be improved. However, with a wider audience of developers, who approached the patch with the perspective of desktop users rather than embedded developers, the patch was significantly improved.

Another great lesson is that, once integrated, the code may be maintained by others. In this case, when the delay calculation code was later modified and moved, the developer who made that subsequent change also moved the preset-LPJ feature automatically. There was no work required on the part of the forum members. The original assumption, that the code would not require much maintenance if left outside the mainline tree, was wrong. By contributing the code, not only was the code itself improved, but the work required to maintain it was reduced.

Another important lesson has to do with the modifications to the feature. During the refinement of the patch, the original option to statically compile the preset value into the kernel was removed. While this change was not the preference of the Bootup Time working group, this made the patch more acceptable to other kernel developers. Since the kernel is used in so many different areas, it is very difficult to balance the needs of all the different developers. In general, it is better to be flexible and accommodate the needs of other developers, even if the eventual result is not exactly what is hoped for. In the case of preset-LPJ, the benefits of improved functionality and reduced maintenance far outweighed the slight reduction in features for the patch. This feature is still one of the most important for being able to reduce bootup time of the Linux kernel, and it is now included in every kernel and distribution of Linux. The feature is available to anyone who uses Linux, whether they are using it in a Consumer Electronics product and whether they are a CELF member or not.

Lessons from rejected patches

The CE Linux Forum has submitted other patches for the Linux kernel that have not been accepted. This is normal. Not every patch submitted to an open source project is accepted. For almost all projects, if a patch is not accepted, then good reasons are given why. If the conventions for formatting the patch were followed, then most likely there will be good, technical reasons why the patch cannot be accepted.

Patches are an expression of requirements. A patch might be an expression of a bug that needs fixing, or a new feature that is needed. Even if a particular patch is rejected, it is useful and important to communicate the need expressed by the patch to other developers. It is very common to find out that there is another solution to the problem, either already existing or under development. If there is no existing solution, ultimately, other developers may respond to the problem. This happens often, although it sometimes takes longer than expected.

Conclusion

If you review the lessons provided in this article for contributing to open source projects, you will notice that they all have to do with benefits to you as a developer, rather than benefits to the community. There are certainly unselfish reasons to contribute to open source, such as a

desire to contribute back to the community from which so much value is received. However, it is important to recognize that there are significant and direct benefits to you as a developer, in participating in open source.

Reasons to contribute to Open Source

- Testing by others
- Bug fixes
- Improvements and extensions
- Reduced maintenance
- Payment back to others for their efforts

(*Only one of these is unselfish!*)

It can be challenging at first to participate in the open source community. However, it gets easier the more you do it, and the rewards for participation are very great

A famous hockey star, Wayne Gretzky, once said "You miss 100 percent of the shots you don't take." The same principle can be applied to making contributions to open source. Contributing to open source will not guarantee that you get results. However, if you don't contribute, you guarantee that you will miss out on the benefits of this powerful development model.