

3: EARLY-STAGE PLANNING

When contemplating a Linux kernel development project, it can be tempting to jump right in and start coding. As with any significant project, though, much of the groundwork for success is best laid before the first line of code is written. Some time spent in early planning and communication can save far more time later on.

3.1: SPECIFYING THE PROBLEM

Like any engineering project, a successful kernel enhancement starts with a clear description of the problem to be solved. In some cases, this step is easy: when a driver is needed for a specific piece of hardware, for example. In others, though, it is tempting to confuse the real problem with the proposed solution, and that can lead to difficulties.

Consider an example: some years ago, developers working with Linux audio sought a way to run applications without dropouts or other artifacts caused by excessive latency in the system. The solution they arrived at was a kernel module intended to hook into the Linux Security Module (LSM) framework; this module could be configured to give specific applications access to the realtime scheduler. This module was implemented and sent to the linux-kernel mailing list, where it immediately ran into problems.

To the audio developers, this security module was sufficient to solve their immediate problem. To the wider kernel community, though, it was seen as a misuse of the LSM framework (which is not intended to confer privileges onto processes which they would not otherwise have) and a risk to system stability. Their preferred solutions involved realtime scheduling access via the rlimit mechanism for the short term, and ongoing latency reduction work in the long term.

The audio community, however, could not see past the particular solution they had implemented; they were unwilling to accept alternatives. The resulting disagreement left those developers feeling disillusioned with the entire kernel development process; one of them went back to an audio list and posted this:

There are a number of very good Linux kernel developers, but they tend to get shouted by a large crowd of arrogant fools. Trying to communicate user requirements to these people is a waste of time. They are much too "intelligent" to listen to lesser mortals.

(<http://lwn.net/Articles/131776/>).

The reality of the situation was different; the kernel developers were far more concerned about system stability, long-term maintenance, and finding the right solution to the problem than they were with a specific module. The moral of the story is to focus on the problem – not a specific solution – and to discuss it with the development community before investing in the creation of a body of code.

So, when contemplating a kernel development project, one should obtain answers to a short set of questions:

3. Early-stage.txt

- What, exactly, is the problem which needs to be solved?
- Who are the users affected by this problem? Which use cases should the solution address?
- How does the kernel fall short in addressing that problem now?

Only then does it make sense to start considering possible solutions.

3.2: EARLY DISCUSSION

When planning a kernel development project, it makes great sense to hold discussions with the community before launching into implementation. Early communication can save time and trouble in a number of ways:

- It may well be that the problem is addressed by the kernel in ways which you have not understood. The Linux kernel is large and has a number of features and capabilities which are not immediately obvious. Not all kernel capabilities are documented as well as one might like, and it is easy to miss things. Your author has seen the posting of a complete driver which duplicated an existing driver that the new author had been unaware of. Code which reinvents existing wheels is not only wasteful; it will also not be accepted into the mainline kernel.
- There may be elements of the proposed solution which will not be acceptable for mainline merging. It is better to find out about problems like this before writing the code.
- It's entirely possible that other developers have thought about the problem; they may have ideas for a better solution, and may be willing to help in the creation of that solution.

Years of experience with the kernel development community have taught a clear lesson: kernel code which is designed and developed behind closed doors invariably has problems which are only revealed when the code is released into the community. Sometimes these problems are severe, requiring months or years of effort before the code can be brought up to the kernel community's standards. Some examples include:

- The Devicescape network stack was designed and implemented for single-processor systems. It could not be merged into the mainline until it was made suitable for multiprocessor systems. Retrofitting locking and such into code is a difficult task; as a result, the merging of this code (now called mac80211) was delayed for over a year.
- The Reiser4 filesystem included a number of capabilities which, in the core kernel developers' opinion, should have been implemented in the virtual filesystem layer instead. It also included features which could not easily be implemented without exposing the system to user-caused deadlocks. The late revelation of these problems - and refusal to address some of them - has caused Reiser4 to stay out of the mainline kernel.
- The AppArmor security module made use of internal virtual filesystem data structures in ways which were considered to be unsafe and

3. Early-stage.txt

unreliable. This code has since been significantly reworked, but remains outside of the mainline.

In each of these cases, a great deal of pain and extra work could have been avoided with some early discussion with the kernel developers.

3.3: WHO DO YOU TALK TO?

When developers decide to take their plans public, the next question will be: where do we start? The answer is to find the right mailing list(s) and the right maintainer. For mailing lists, the best approach is to look in the MAINTAINERS file for a relevant place to post. If there is a suitable subsystem list, posting there is often preferable to posting on linux-kernel; you are more likely to reach developers with expertise in the relevant subsystem and the environment may be more supportive.

Finding maintainers can be a bit harder. Again, the MAINTAINERS file is the place to start. That file tends to not always be up to date, though, and not all subsystems are represented there. The person listed in the MAINTAINERS file may, in fact, not be the person who is actually acting in that role currently. So, when there is doubt about who to contact, a useful trick is to use git (and "git log" in particular) to see who is currently active within the subsystem of interest. Look at who is writing patches, and who, if anybody, is attaching Signed-off-by lines to those patches. Those are the people who will be best placed to help with a new development project.

If all else fails, talking to Andrew Morton can be an effective way to track down a maintainer for a specific piece of code.

3.4: WHEN TO POST?

If possible, posting your plans during the early stages can only be helpful. Describe the problem being solved and any plans that have been made on how the implementation will be done. Any information you can provide can help the development community provide useful input on the project.

One discouraging thing which can happen at this stage is not a hostile reaction, but, instead, little or no reaction at all. The sad truth of the matter is (1) kernel developers tend to be busy, (2) there is no shortage of people with grand plans and little code (or even prospect of code) to back them up, and (3) nobody is obligated to review or comment on ideas posted by others. If a request-for-comments posting yields little in the way of comments, do not assume that it means there is no interest in the project. Unfortunately, you also cannot assume that there are no problems with your idea. The best thing to do in this situation is to proceed, keeping the community informed as you go.

3.5: GETTING OFFICIAL BUY-IN

If your work is being done in a corporate environment - as most Linux kernel work is - you must, obviously, have permission from suitably

3. Early-stage.txt

empowered managers before you can post your company's plans or code to a public mailing list. The posting of code which has not been cleared for release under a GPL-compatible license can be especially problematic; the sooner that a company's management and legal staff can agree on the posting of a kernel development project, the better off everybody involved will be.

Some readers may be thinking at this point that their kernel work is intended to support a product which does not yet have an officially acknowledged existence. Revealing their employer's plans on a public mailing list may not be a viable option. In cases like this, it is worth considering whether the secrecy is really necessary; there is often no real need to keep development plans behind closed doors.

That said, there are also cases where a company legitimately cannot disclose its plans early in the development process. Companies with experienced kernel developers may choose to proceed in an open-loop manner on the assumption that they will be able to avoid serious integration problems later. For companies without that sort of in-house expertise, the best option is often to hire an outside developer to review the plans under a non-disclosure agreement. The Linux Foundation operates an NDA program designed to help with this sort of situation; more information can be found at:

http://www.linuxfoundation.org/en/NDA_program

This kind of review is often enough to avoid serious problems later on without requiring public disclosure of the project.