Professional

Linux® Kernel Architecture

Wolfgang Mauerer

WILEY

Wiley Publishing, Inc.

```
(d) While I have reviewed the patch and believe it to be sound, I
    do not (unless explicitly stated elsewhere) make any
    warranties or guarantees that it will achieve its stated
    purpose or function properly in any given situation.
```

Another new tag introduced in this context is `Tested-by`, which — you guessed it — states that the patch has been tested by the signer, and that the test has left enough of the machine to add a `Tested-by` tag to the patch.

F.4 Linux and Academia

Writing an operating system is not an easy task — as I'm sure you'll agree, it is one of the most involved challenges for software engineers. Many of the developers participating in the creation of the Linux kernel are among the most knowledgeable in their field, given that Linux is one of the best operating systems available. Academic degrees are not uncommon among developers, and computer science degrees are surely not underrepresented degrees.[5]

Operating systems are also the subject of active academic research. As with every other research field, there's a certain amount of theory that goes along with OS research, and this is just natural — you cannot tackle all problems in a practical way. In contrast to many other research areas that are concerned with fundamental problems, however, OS research works on inherently practical problems, and should therefore have an impact on practical things. What is OS research good for if it does not help to improve operating systems? And because an operating system is an inherently practical product (who, after all, would need a *theoretical* operating system? Hypothetical computers certainly have no use for an operating system, and even less do real computers require a theoretical OS), the outcome of OS research *has* to influence practice. People working on loop quantum gravity might be exempted from having to consider the practical impact of their work, but this is certainly not the case for OS research.

With this in mind, one could expect that Linux and the academic community are closely associated, but unfortunately, this is not the case. Quoting academic work in the kernel sources is a rare occurrence, and seeing the kernel being quoted in research papers is also not something that happens every day.

This is especially astonishing because the academic world used to have a close affiliation with UNIX, particularly with the Berkeley System Distribution (BSD) family. It's fair to say that BSD *is* the product of academic research, and for a long time, academia was the driving force behind this project.

A study published by the Linux Foundation [KHCM] has shown that contributions from academia account for 0.8 percent of all changes in recent kernel versions. Considering that a large number of ideas circulate in the academic community, this ratio is astonishingly low, and it would be worthwhile to improve the situation — for the benefit of both the kernel *and* academia. Open source is all about sharing things, and sharing good ideas is also a worthy goal.

Linux had a slightly bumpy start in its relations with academia. One of Linus Torvalds's initial motivations to write Linux was his dissatisfaction with Minix, a simple teaching operating system designed to educate students. This led to a famous debate between Torvalds and Andrew Tanenbaum, the creator of Minix. Tanenbaum suggested that Linux was obsolete because its design would not comply with

[5]Notice that I did not perform any quantitative analysis on this, but the curricula vitae of many developers are readily available on the Internet that support this suspicion (as does common sense.)

what the academic world envisioned to be suitable for future operating systems, and his arguments were collected in a Usenet newsgroup posting titled "Linux is obsolete." This, naturally, caused Linus Torvalds to reply, and one of his statements was the following:

```
Re 2: your job is being a professor and researcher: That's one hell of a
good excuse for some of the brain-damages of minix.
```

Although it was soon admitted that the message was a little rash, it reflects the attitude that is sometimes displayed by the kernel community toward academic research. Real-world operating systems and OS research are perceived as things that don't quite fit together.

This may indeed be true sometimes: Much academic research is not *supposed* to be integrated into real-world products, especially when it is concerned with fundamental issues. But as mentioned previously, there are also practical components of research, and these could often help to improve the kernel. Unfortunately, OS researchers and OS implementors have somewhat lost connection with each other, and Rob Pike, a member of the former UNIX team at Bell Labs, has gone so far as to make the pessimistic claim that systems software research is irrelevant.[6]

Contributing code to the kernel is hard for researchers for many reasons, one of which is that they have to take many different operating systems into account. It is already hard to keep up with the pace of Linux kernel development, but it is virtually impossible to chase all important operating systems in use today. Therefore, researchers usually cannot provide more than proof-of-concept implementations of their ideas. Integrating these into the kernel requires some effort from both communities. Consider, for instance, the integration of the swap token mechanism into the kernel. This was proposed in research as discussed in the next section, but has been implemented for the kernel by Rik van Riel, a kernel developer working in the area of memory management. The approach has proved to be quite successful, and could well serve as a role model for further collaboration.

Interaction between both communities is complicated by the following two aspects of kernel development:

- Many developers do not consider proposals without concrete code, and refuse to discuss the issue any further.
- Even if code is submitted to the mailing lists, a good part of the work will start only after the initial submission. Adaption of proposed code to a specific system is not highly credited in academia, so researchers have a natural tendency to avoid this step.

Ultimately, this leads to the conclusion that the interface between kernel development and academic research ideally requires one individual from each side collaborating with each other. If this is not possible, then it is a definitive advantage and surely worth the effort if researchers try to adapt to the culture of kernel development as much as possible.

F.4.1 Some Examples

This section presents some examples of when research results have been turned into kernel code and could help to improve particular aspects of Linux. Note that the presented selection is naturally not

[6] See `www.cs.bell-labs.com/who/rob/utah2000.pdf`. Since Pike also claims that the only progress in the operating system area comes from Microsoft, I certainly don't believe all his claims, but the talk nevertheless contains many noteworthy and valid ideas.

comprehensive, and the impact of academic research would be really negligible if it ever could be. It is primarily used to highlight that both worlds *can* benefit from each other.

- The swap token as discussed in Chapter 18 was first described in the paper "Token-Ordered LRU: An Effective Replacement Policy and its Implementation in Linux Systems" by S. Jiang and X. Zhang (Performance Evaluation, Vol. 60, Issue 1–4, 2005). Subsequently, it was implemented in kernel 2.6.9 by Rik van Riel. Interestingly, the paper extended kernel 2.2.14 to demonstrate the usefulness of the approach, but the corresponding code was never included in the mainline kernel.
- The slab allocator as discussed in Chapter 3 is directly based on a paper that describes the implementation of the slab system in Solaris: "The Slab Allocator: An Object-Caching Kernel Memory Allocator," Proceedings of the Summer 1994 USENIX Conference.
- The techniques of the anticipatory I/O scheduler (which was mentioned in Chapter 6, but not discussed in detail) were first presented in "Anticipatory Scheduling: A Disk Scheduling Framework to Overcome Deceptive Idleness in Synchronous I/O," 18th ACM Symposium on Operating Systems Principles, 2001.
- As discussed in Chapter 18, Linux employs a variant of the least-recently used technique to identify active pages and distinguish them from inactive pages. The paper "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement" by S. Jiang, F. Chen, and X. Zhang (Proceedings of 2005 USENIX Annual Technical Conference) describes a page-replacement algorithm that not only prioritizes pages based on the time of their last access, but also incorporates the frequency with which pages are accessed. Patches have been designed by Rik van Riel and Peter Zijlstra, and the method has also been considered as a possible merge candidate (see `www.lwn.net/Articles/147879/`). The reason why you have read nothing about this technique in the preceeding chapters is simple: The patches have not yet made it into mainline. They are, however, examples of how Linux developers do sometimes actively try to integrate research results into the kernel.

The ideas presented in these papers have been directly integrated into the Linux kernel as direct extensions of existing code. Some examples of older papers that have had an indirect influence on the kernel include the following:

- The generic structure of the block layer that acts as a level of indirection between filesystems and disks is described in "The Logical Disk: A New Approach to Improving File Systems," by W. de Jonge, M. F. Kaashoeck, and W. C. Hsieh. Essentially, it describes techniques to decouple blocks on physical disks from logical disks as observed by the operating system, and this builds the fundament for the logical volume manager and the device mapper.
- Many key concepts of the extended filesystem family originate from other filesystems, and one particular example is the paper "A Fast File System for UNIX" by M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry (ACM Transactions on Computer Systems, 1984). It describes the use of multiple possible block sizes on disk, and introduces the idea of mapping a logical sequence of data to a sequential series of blocks on disk.

Tracking the indirect influence of older papers is naturally much harder than seeing ideas from research being directly integrated. The more generic an idea is, the more ubiquitous it will become if it prevails, and the harder it becomes to recognize the idea as such. At some point, it will have been absorbed into the field, and be indistinguishable from common knowledge. Or would you deem it necessary to quote any paper on the fact that computers tend to work with binary digits?

Essentially, most core ideas of the UNIX operating system are also present in Linux. Many of these ideas are today ubiquitous, but were new at the time UNIX was invented. This includes, for instance, the idea that nearly everything can be represented by a file as discussed in Chapter 8. Namespaces are another example for a technology that does indirectly stem from academic research: They were introduced as an integral part of Plan 9 — the successor to UNIX co-developed by some of the inventors of UNIX — many years before they were adopted into the mainline kernel.[7] The `/proc` filesystem is also modeled by the example of Plan 9.

Many other fundamental ideas of UNIX appear as integral parts of Linux without being recognized as research results, but this is not the direct concern of this section. However, it is interesting to observe where many concepts of Linux have their roots, such as in Vahalia's highly recommended technical discussion of UNIX internals for many flavors of the system [Vah96]. The account by Salus [Sal94] illuminates the history of UNIX, and allows for understanding why many things are designed the way they are.

F.4.2 Adopting Research

The preceding examples demonstrate that it *is* possible to integrate research results with the Linux kernel. But considering the magnitude of OS research, and the number of results integrated into the kernel, there seem to be some obstacles to transferring results from one world into another. One essential factor is that each community functions quite differently from each other. To my knowledge this point has not received the consideration it deserves (at least not in writing); therefore, this section highlights some of the essential differences.

Notice that the kernel sources contain some interesting information on how the kernel developers deal with project management issues in `Documentation/ManagementStyle`. The document also addresses some of the questions discussed here.

Different Communities

Software development and OS research seem to be dry and purely technical to many people, but both have an enormous social component: The acceptance of any work is based on its acceptance in the community, which is nothing else than acceptance by individual developers and researchers. This requires that individuals judge the contributions of other individuals, and as most readers will agree, this is always a difficult thing in a world of colorful, different, and sometimes complicated characters. In an ideal world, judgment would be solely based on objective criteria, but this is not the case in reality: People are only human, and sympathy, personal tastes, acquaintances, dislikes, bias, and the ability to communicate with each other play a crucial role.

One approach to this problem is to simply ignore it — pretend that we live in an ideal world where judgment *is* done on a purely technical and objective level, and all problems automatically disappear. This solution is adopted astonishingly often, especially in "official" statements.

[7] Notice that Plan 9 was not developed at a "classical" academic institution, but at the research division of Bell Labs, which is nowadays affiliated with Lucent Technologies. However, the methodology used is very similar to that of academic institutions: Papers are published about Plan 9, talks are held, and conferences are organized. Therefore, this appendix subsumes it under the same category as academia. The web site `cm.bell-labs.com/plan9` contains more information about Plan 9.

But even if the problem is acknowledged, it is not easy to solve. Consider how decisions are often made in the research community to decide if a work is worthwhile (and should be credited by being admitted to a conference, or published in a paper) or not:

1. After research results have (hopefully) been obtained, they are written up in a paper and submitted to a journal (or conference, or similar, but this discussion will focus on publication for simplicity's sake).
2. The paper is submitted to one or more referees who have to evaluate the work. They have to judge correctness, validity, and scientific importance, and can point to weaknesses or things that should be improved. Usually reviewers are anonymous, and should not be directly related with the author personally or professionally.
3. Depending on the referee's appraisal, the editor can decide to reject or accept the paper. In the latter case, the editor may require the author to incorporate improvements suggested by the referees. Another round of peer review may take place after the improvements have been made.

Usually, the identity of authors is known to the referee, but not vice versa.

Work is considered worthwhile in the kernel community if it is included into some official tree. The way to achieve such an inclusion goes essentially along these lines:

- Code is submitted to an appropriate mailing list.
- Everyone on the mailing list can request changes to the code, and desired improvements are discussed in public.
- The code is adapted to the desires of the community. This can be tricky because there are often orthogonal opinions as to what constitutes an improvement and what will deteriorate the code.
- The code is re-submitted, and discussion starts anew.
- Once the code has achieved the desired form and a consensus is reached, it is integrated into official trees.

Notice that it is possible for people with high and long-standing reputations in their fields (which is, again, a social factor) to shortcut the process in both areas, but these cases are not of interest here.

There are similarities between the academic and kernel development communities, and both have their strengths and weaknesses. For example, there are some important differences between the review process in each community:

- Reviewing code for the kernel is not a formalized process, and there is no central authority to initiate code review. Review is performed completely voluntarily and uncoordinated — if no one is interested in the submitted code, the mailing lists can remain silent.

  Although review in the academic world is usually also performed voluntarily and without payment, it is impossible for submissions to be completely ignored. Papers are guaranteed to get *some* feedback, although it can be very superficial.
- The identities of the submitter and reviewer are known to each other in the kernel world, and both can interact directly. In the academic world, this is usually not the case, and conversation

between the author and reviewer is mediated by the editor. Additionally, only a few rounds of exchanging arguments between the author and reviewers are possible before the editor decides to either accept or reject a submission.

- The result of a review is only known to the submitter, referees, and editor in the academic world. Usually the whole review process is public in the kernel world, and can be read by everyone.

In both worlds, reviewers can pass harsh criticism to the submitter. In the academic world, formulations in the inverse direction are usually chosen with much more care, while this depends on the identity of the submitter and reviewer in the kernel world.

Critique is certainly valuable and essential to improve the quality of any work, but *receiving* critique is a complicated matter. How this is handled is another important difference between kernel development and the academic world.

Harassing people verbally in various creative, and often insulting, ways has become a trademark of some kernel developers — and the corresponding statements are publically available on the Internet. This poses a serious problem, because nobody likes to be insulted in public, and developers can be driven away by this fairly quickly. This concern is shared by several leading Linux developers, but because all people on the mailing lists are grown-ups, it is not possible to solve this problem in any other form than appealing for more fairness, which is not always accepted.

Receiving a harsh critique by an anonymous referee in the academic world is certainly not particularly enjoyable, but it is much easier to be accused of having failed in private than in public.

As you can see from the following documentation excerpt, kernel developers do *not* strive for complete political correctness as a means of solving this problem:

Documentation/ManagementStyle

```
The option of being unfailingly polite really doesn't exist. Nobody will
trust somebody who is so clearly hiding his true character.
```

Pulling each other's legs *can* be a good thing, and is something of an intellectual challenge when properly employed. But it's also very easy to overdo it and end up with insulting accusations, which nobody likes to receive but unfortunately, everyone should be prepared for in the kernel world.

While the review process of the kernel world can be considerably more challenging socially than the academic counterpart, it also tends to be much more effective, taken that people are not driven away by the approach: Patches on the kernel mailing list usually go through many iterations before they are deemed to be acceptable, and in each iteration step remaining problems are identified by reviewers and can be changed by the authors. Because the goal of the kernel is to be the best in the world, it is important that only really good code is integrated. Such code is usually not created from the very beginning, but only after a period of improvement and refinement. The whole point of the review process is to generate the best possible code, and this often succeeds in practice.

The effect of review on academic papers is usually different. If a submission is rejected by one journal, it will certainly be revised by the authors to address the shortcomings. However, readers are invited to judge on their own how big the probability is that *really* substantial revisions are made considering that on the one hand, there is considerable pressure to publish as many papers as possible for gaining scientific reputation, and on the other hand, there are a large number of different (possibly less-renowned) journals to which the work can alternatively be submitted — and that these journals *rely* as their economic

foundation on submissions by authors who pay(!) for being published. This is different with kernel code: Either you get code into the kernel, or a considerable amount of effort has been wasted.[8] This naturally provides a large incentive to put effort into improving the code.

Although the approaches employed by the academic and kernel communities to assess and ensure the quality of submissions are similar at first glance, there are numerous differences between them. Different cultures can be a considerable barrier for the exchange of ideas and code between both worlds, and should be taken into account when it comes to a collaboration between kernel and academia.

F.5 Summary

As one of the largest open source efforts in the world, the Linux kernel is not just interesting from a technological perspective, but also because a novel and unique way of distributed development across the whole world and between otherwise competing companies is employed. This appendix described how the process is organized, and what the requirements for contribution are. It also analyzed the connection between kernel development and academic research. In this appendix, you learned how the two worlds interact, how differences can arise from different "cultures," and how these are best bridged.

[8] It is surely possible to maintain code out-of-tree, and this has proven useful in many cases, but the final and most rewarding goal for developers (and their employers!) is nevertheless to get work into the mainline kernel.