Charles University in Prague
Faculty of Mathematics and Physics

ERRATA AND COMMENTS

Martin Děcký

Application of Software Components in Operating System Design

Department of Distributed and Dependable Systems

Prague 2015

1 Errata

Since the submission of the doctoral thesis titled *Application of Software Components in Operating System Design* in June 2015, several mistakes have been found in the original text – either by the reviewers of the thesis (Björn Döbel, Michal Sojka) or by me as the author of the thesis (Martin Děcký).

The vast majority of the mistakes are simple typos, grammatical errors and stylistic omissions. I do not provide a list of these minor mistakes here, because their corrections do not change the intended meaning of the text of the thesis. If necessary, a complete list of all corrections can be always generated from the version control system where the source text of my doctoral thesis is kept.[1]

The updated version of the doctoral thesis with corrections and other updates is available for download at `http://d3s.mff.cuni.cz/~decky/phd/thesis_updated.pdf`

What follows is a commented list of errata corrections that actually change the intended meaning of the text of the thesis:

Section 2.2, page 6 A forgotten reference to "[17]" has been added to the paragraph describing the publication *Resource Sharing in Performance Models*.

Chapter 4, page 20 A factual historical error has been corrected. The microkernel developer room has been organized at FOSDEM since 2012, not since 2011.

Section 5.1.6, page 28 A new paragraph has been added that explains why Chapter 5 does not provide traditional citation references to the mentioned operating system projects. The references are being maintained separately in the form of a dedicated wiki page `http://trac.helenos.org/wiki/ZOO` which can be easily kept up-to-date.

Section 6.2.1, page 55 The incorrect formulation that the kernel of HelenOS provides "constant-sized kernel message queues" has been replaced by the formulation that the kernel of HelenOS provides "bounded kernel message queues".

Section 8.2.3, page 83 Similarly to the previous case, the formulation has been changed from "fixed-size kernel dispatch buffers" to "bounded kernel dispatch buffers".

Entire text The numbers of successfully defended HelenOS-related bachelor and master theses have been updated throughout the entire text to reflect the state as of September 2015. Also references "[39]" and "[44]" have been added based on the suggestions of the reviewers.

[1] `svn://svn.decky.cz/decky.cz/phd`

2 Comments on Reviews

I would like to express my gratitude to the official reviewers of my thesis (Björn Döbel and Michal Sojka) for providing their helpful comments and constructive criticism. The following paragraphs contain my concise comments on the criticism and replies to the questions asked by the reviewers.

2.1 Lack of Performance Evaluation

Björn Döbel writes in his review:

> [...] *the thesis completely lacks a traditional evaluation of quantitative properties. It is a fundamental principle of operating systems design to evaluate the results of one's work using standard benchmark programs and compare those results to other systems.*

Michal Sojka requests a similar kind of evaluation in his review:

> [...] *HelenOS uses advanced algorithms and data structures but what is missing is the evaluation of whether the benefits of the their use correspond to the expectation or at least to the results found in the literature. The question is how are these things evaluated and how well is HelenOS doing compared to other mainstream or microkernel-based OSes. I would be interested in things like performance and scalability on real world use-cases.*

The thesis deliberately focused on the overall development methodology of HelenOS and not on individual implementation features of HelenOS and their performance.

That being written, the performance evaluation of several individual implementation features of HelenOS has been done and has been published previously, mostly using microbenchmarks. For instance, the IPC performance in HelenOS has been evaluated in the master thesis of Vojtěch Horký [9] and our custom Read-Copy-Update algorithm and the Concurrent Hash Table have been thoroughly evaluated in the master thesis of Adam Hraška [10].

However, I must acknowledge that while the text of my thesis references these sources, it might have put more stress on the fact that the respective referenced theses indeed contain these performance evaluations.

Finally, the text of my thesis clearly describes where are currently the major performance bottlenecks of HelenOS (see the Section 9.3.1.2 and the Section 10.1.2). In my opinion, until these bottlenecks receive proper development attention, running synthetic benchmarks on HelenOS in order to compare its performance with other operating systems is a waste of time. We can predict the results of such comparison and our prediction is currently not favorable to HelenOS.

Björn Döbel further writes:

> [...] *Mr. Děcký argues that traditional microkernel systems strictly avoid implementing any complex data structures or algorithms within the kernel and require those to be moved to user-level components. He argues that HelenOS forgoes this strictness and uses complex features, such as concurrent hash tables to improve kernel resource management. Unfortunately, this argument is made purely from a philosophical standpoint. The argument would have been much more convincing if it was backed with some kind of benchmark that shows the improvements such complex algorithms provide.*

The argument is not only philosophical, but it is based on the evaluation of concurrent algorithms and data structures done in the past by others, in the context of other operating systems (for instance [2, 3, 11, 16]). We use these results as reasonable indications that exploring the potential of such advanced algorithms and data structures might be beneficial also in the context of HelenOS.

Furthermore, the microbenchmarks of our own implementations support our initial expectations. To quote explicitly from the conclusion of [10]:

> CHT [Concurrent Hash Table] successfully replaced the original kernel futex subsystem design which resulted in dramatic scalability improvements and even reduced the subsystem's singlethreaded base cost. [...] The presented results demonstrate that the new futex subsystem is linearly scalable [...] Last but not least, we determined that RCU [Read-Copy-Update] allowed us to considerably speed up fibril locking of singlethreaded user programs.

The raw measurements and original detailed analysis can be found in [10]. Again, I acknowledge that the text of my thesis should have pointed to this work more explicitly and it should have stressed the performance evaluation done, but at time of writing I unfortunately did not see it in this way.

As the next point of criticism, Björn Döbel writes:

> [...] *the thesis also does a poor job in quantitatively analyzing the system's reliability. Mr. Děcký duly notes that each of the validation activities incorporated into HelenOS build process was able to pinpoint bugs that might have otherwise gone unnoticed. Unfortunately, this usually only mentions a number of change sets to the HelenOS system, but does not give details on the number of bugs found in the system and how these numbers relate to other operating systems.*

In hindsight, I must acknowledge that such quantitative evaluation would have been indeed interesting, but also quite challenging, because it would probably require some kind of methodology that would be able to normalize different types of bugs in very different instances of operating systems (differing in their size, architecture, number of contributors, etc.). For the results to be really convincing, some kind of controlled evaluation experiment would be also necessary.

To provide at least a very limited answer: Since August 2009, the developers of HelenOS have fixed approximately 300 bugs in the mainline branch of HelenOS. Approximately 15 of these bugs were detected by static analyzers and verifiers, which accounts for about 5 % of bugs. However, this classification is completely missing bugs that were fixed by the developers in their own working source trees even before any faulty code was committed (analyzing these bugs would require some kind of data collection that cannot be possibly done retroactively). For example, some of the bugs might have been detected by the verifying compiler (the code was not even compilable – which is actually the best way of integrating verification into the development process, as noted in the text of the thesis). Some of the bugs might have been detected by regression tests, also before merging into the mainline branch.

However, I see this as a very constructive and inspiring point of criticism and I declare this an important part of our future work on the verification of HelenOS.

Finally, Björn Döbel comments:

> [...] *using those* [verification] *tools comes at a price in terms of development effort (someone has to add the respective annotations etc. to code), compile time (how long do those tools run?), and perhaps runtime performance. The thesis unfortunately does not quantify these costs in order to relate them to other approaches.*

The reason why the thesis focuses more on qualitative than quantitative evaluation is explained in the conclusion: "[...] designing and implementing an entire operating system is not a trivial task and even with the manpower of all the contributors we do not have the luxury of being able to compare multiple independent and complete implementations of HelenOS based on different designs."

The same observation unfortunately applies also to any quantitative evaluation of the costs of our verification approaches. The costs of amending the source code with verification annotations and running the verification tools were sunken into the costs of the other development efforts. They cannot be separated retroactively because we have not collected the data originally and we do not have a second version of HelenOS implemented without this additional overhead that we might use for comparison.

Similarly to the previous point of criticism, it is possible to provide some rough values related to the run-time overhead: The overhead of the utilization of verifying compilers is negligeable in terms of total compilation time. The utilization of static analyzers and verifiers represents a typical overhead of tens of seconds per each run. Our suite of continuous integration and regression tests represents a typical overhead of tens of minutes per each run for our reproducible builds and potentially a couple of hours for all permissible configurations.

But again, I understand this point of criticism as very inspiring for our future work. In my opinion, it calls for a full-fledged controlled experiment in which the trade-offs of utilizing and not utilizing our verification methods would be rigorously evaluated.

2.2 Partial Lack of Technical Detail

Björn Döbel writes in his review:

> *Chapter 6 gives a very brief (4 page) tour of HelenOS' kernel features, while Chapter 7 spends 12 pages discussing HelenOS' development process. Being an operating systems person I would have enjoyed the ratio to be the other way round* [...]

A similar comment is also expressed by Michal Sojka. I deliberately did not go into many technical implementation details in the text of the thesis, because the focus of the thesis was indeed on the overall development methodology and not on individual implementation features. Whenever the thesis describes some implementation detail, it is intended as an illustration of the other aspects (design, development process, verification).

No specific implementation work has been done solely for the purpose of writing this thesis. Every implementation effort ever undertaken was done for the general benefits of HelenOS. Therefore I believe that the goal of the thesis was rightfully not to replace or duplicate the existing technical documentation of HelenOS and previously published theses that discuss individual implementation aspects in much detail [13].

In other words, the text of the thesis omits some technical details not because I would want to hide something (after all, the entire source code of HelenOS is public), but to avoid reiterating something that is already described elsewhere.

Björn Döbel also adds:

> *Section 8.3.3.1 spends less than half a page on describing a byte-code interpretation mechanism that allows the kernel to execute user-provided device driver code in kernel mode.* [...] *the idea of protecting the kernel from this unsafe code by using a byte code interpreter sounds pretty novel and in the position of the author I would have put much more focus on this topic* [...]

This feature of the HelenOS kernel has been originally implemented by Ondřej Palkovský in 2005 as a fairly straightforward response to the need to deassert level-triggered interrupts (within the kernel interrupt handler context) caused by devices managed by user space drivers. The feature has been gradually extended over time to support basic conditional evaluation while naturally respecting the requirement that the untrusted user space device driver cannot be allowed to do any harm in the kernel space via the IRQ byte-code.

The bulk of the byte-code processing is implemented in `kernel/generic/src/ipc/irq.c` in the HelenOS source tree. The byte-code is register-based with six GPR-sized registers (called "scratch memory"). The byte-code instructions are as follows:

`CMD_PIO_READ_{8|16|32}` Read a value from a fixed physical memory location (memory-mapped I/O) or I/O address space to a scratch register.

`CMD_PIO_WRITE_{8|16|32}` Write a constant value to a fixed physical memory location (memory-mapped I/O) or I/O address space.

`CMD_PIO_WRITE_A_{8|16|32}` Write a value from a scratch register to a fixed physical memory location (memory-mapped I/O) or I/O address space.

`CMD_LOAD` Load a constant to a scratch register.

`CMD_AND` Compute a bitwise conjunction between a constant and a scratch register value.

`CMD_PREDICATE` If the value in the given scratch register is zero, skip ahead a constant number of instructions.

`CMD_ACCEPT` Accept processing of the interrupt request in user space.

`CMD_DECLINE` Decline processing of the interrupt request (implicit action).

As can be trivially observed, the byte-code instructions do not operate with computed addresses (both in case of the I/O addresses and scratch registers), therefore the safety of these addresses can be statically checked. Furthermore, the execution control flow can change only as a result of the values in scratch registers and only by skipping a few byte-code instructions. This guarantees that the execution time of the entire byte-code routine is always bounded.

Despite its simplicity and extremely limited expressive power, our byte-code is sufficient for all device drivers currently implemented in HelenOS for all supported architectures. If necessary, it could be expanded in the future, while possibly keeping its safety properties intact.

The reason why a similar technical description is missing from the text of the thesis is that I have considered our IRQ byte-code interpreter a completely straightforward engineering solution to the problem at hand.

A much more elaborate implementation of the same ideas has been designed and implemented by Lukáš Lipavský for his master thesis [15] in the context of GNU/Linux (I have supervised the thesis and I have pointed Lukáš Lipavský to the HelenOS implementation for inspiration), where a slightly more powerful safe byte-code is compiled from a C-like source code snippet (instead of being written by hand).

2.3 Lack of Citations

Björn Döbel writes in his review:

> *Scientific style demands using citations to allow the reader to get pointers to other work that is discussed in the context of a scientific publication. While Mr. Děcký in general adheres to common standards, I would have preferred a much higher rate and amount of citations. Especially, when talking about related work from the operating systems area, Mr. Děcký names lots of projects* [...] *without giving any pointers to their sources, be it scientific publications or rather web pages.*

The decision to refer to some operating systems and projects just by their names was simply a pragmatic one. The discussion of the related work talks not only about academic projects where there is usually one "canonical" publication to point to (although the notion of a "canonical" publication is somewhat questionable), but it talks also about commercial and hobby projects. It also includes projects that have been defunct for a long time. Especially in the latter cases, an URL in a printed text pointing to a source can be soon obsolete. And, frankly, I do not know what should be the canonical reference to **OS/2**, **BeOS** or **SkyOS**.

Björn Döbel suggests, for example, to use the technical report by Feske and Helmuth [6] as the proper reference to **Genode**. I must disagree with the factual correctness of such reference, since the Genode platform has moved far beyond the original Bastei architecture proposal. On the other hand, I believe that the plain names of the discussed operating systems are unique enough to work as identifiers.

However, I acknowledge that providing a more durable and comprehensive list of pointers and references might be beneficial for any future reader of the text. Therefore I have created a dedicated page on the HelenOS wiki where pointers and references to other operating systems and projects could be maintained in the long run: `http://trac.helenos.org/wiki/ZOO`

Hereby I thank Björn Döbel for suggesting some references of his own. These have been already included in the wiki page. Also please note that some of the suggested references have been actually used in the thesis.

Finally, Björn Döbel adds to his previous comment:

> *The same applies to sections where criticism of microkernels is countered but no reference to any such criticism is provided.*

While I would like to strongly stress that the parts of the thesis that talk about specific criticism of microkernels do provide references (specifically [1, 17]), I acknowledge that providing even more references is indeed possible.

I thank Björn Döbel for providing a reference of his own ([7]) which has been incorporated into the updated text of the thesis. On the other hand, this particular paper has been strongly criticized by the microkernel community as inaccurate [8].

2.4 Real-Time

Michal Sojka writes in his review:

> *It is unfortunate that real-time capabilities were explicitly excluded as being too specific to a single purpose. It has been shown in 2004 that a general-purpose OS (Linux) can be made a real-time OS at the same time (thanks to the preempt-rt patch). Nowadays, there is big industry demand for verified real-time operating systems and HelenOS could be more relevant if it provides real-time features.*

I believe that this interpretation (exclusion of real-time capabilities) is a misunderstanding of what I intended to express in the thesis. Let me clarify this explicitly. The general-purpose design principle of HelenOS states that:

- The design of the operating system should not seek excellence in any particular area by sacrificing generality and reasonable fitness for any other purpose.

It is important to understand that this design principle does not rule out the possibility to target specific particular areas. It only requires to do so in such a way that would not be detrimental to the generality of HelenOS. As a matter of fact, real-time features in HelenOS have been seriously planned by the HelenOS community at least since 2013 [18] and this has been the primary motivation for my personal participation in other real-time related projects [5, 14] even long before that.

2.5 Fixed-size Kernel Dispatch Buffers

Michal Sojka asks the following question in his review:

> *While advanced data structures are used at many places, IPC uses "fixed-size kernel dispatch buffers". I tried to look up the corresponding code, but I only found places with memory allocation and linked lists in the IPC path. Could the author clarify the use of fixed-size buffers and what is the initial or optimal size of these buffers?*

The formulation in the original text of the thesis was indeed quite inaccurate in this matter. The correct explanation is that the kernel IPC dispatch buffers are not of a fixed size, but their size is bounded by a fixed compile-time constant. The buffers for the asynchronous messages are still allocated dynamically in order to ensure optimal memory utilization, but the upper bound guarantees that no user space component consumes more than a predefined amount of kernel memory.

The upper bound is enforced by the `check_call_limit()` function in `kernel/generic/src/ipc/sysipc.c` in the HelenOS source tree. The actual bound is defined by the `IPC_MAX_ASYNC_CALLS` constant declared in `abi/include/abi/ipc/ipc.h`. The current value of this constant is 4.

2.6 Dynamic Priorities

Michal Sojka asks the following question in his review:

> *HelenOS kernel "does not support any kind of static priorities", but there are several run queues with different priorities in the scheduler. How are these priorities used?*

The HelenOS scheduler (the source code can be found in the HelenOS source tree in `kernel/generic/src/proc/scheduler.c`) implements a variant of the traditional multilevel feedback round-robin preemptive scheduler [4]. This scheduler has a fixed number of FIFO queues (16 in the case of HelenOS) of schedulable threads. Each queue in this arrangement represents one dynamic priority level.

The scheduler always selects a ready thread from the head of the highest non-empty priority level. Newly created threads are enqueued in the highest priority level. Threads that voluntarily relinquish their time quantum are enqueued in the same priority level they were previously scheduled from. Threads that need to be preempted forcefully at the end of their scheduling quantum are enqueued in the priority level just below their original level and blocked threads are enqueued in the priority level just above their original level.

A slightly more detailed description of the HelenOS scheduler implementation can be found in the original HelenOS design documentation [12]. Our scheduler design has dynamic priorities that are assigned by the scheduler according to the dynamic behavior of the current workload. However, it lacks static priorities (priority classes, niceness factor, etc.) that would define a strict user-configurable ordering of the priorities of the threads. Also the scheduling time quanta are currently constant and they are not individually configurable for each thread.

References

[1] Ballesteros F. J., Fernandez L. L.: *The Network Hardware Is the Operating System*, in the Proceedings of the 6th Workshop on Hot Topics in Operating Systems, IEEE, 1997

[2] Bonwick J.: *The Slab Allocator: An Object-Caching Kernel Memory Allocator*, in the Proceedings of USENIX Summer 1994 Technical Conference, USENIX Association, 1994

[3] Bonwick J., Adams J.: *Magazines and Vmem: Extending the Slab Allocator to Many CPUs and Arbitrary Resources*, in the Proceedings of the General Track, USENIX Annual Technical Conference, USENIX Association, 2001

[4] Corbató F. J., Daggett M. M., Daley R. C.: *An Experimental Time-Sharing System*, in the Proceedings of the Spring Joint Computer Conference 21, ACM, 1962

[5] Děcký M.: *Real-Time Java Assessment Technical Note 1 Appendix – Predictability and Performance Benchmarking*, SciSys UK Ltd., European Space Agency, 2008

[6] Feske N., Helmuth C.: *Design of the Bastei OS Architecture*, Technical Report, TU Dresden 2006

[7] Hand S., Warfield A., Fraser K., Kotsovinos E., Magenheimer D.: *Are Virtual Machine Monitors Microkernels Done Right?*, in the Proceedings of the 10th USENIX Conference on Hot Topics in Operating Systems, ACM, 2005

[8] Heiser G., Uhlig V., LeVasseur J.: *Are Virtual Machine Monitors Microkernels Done Right?*, in ACM SIGOPS Operating Systems Review, Volume 40, Issue 1, ACM, 2006

[9] Horký V.: *Support for NUMA hardware in HelenOS*, Master Thesis, Charles University in Prague, 2011

[10] Hraška A.: *Read-Copy-Update for HelenOS*, Master Thesis, Charles University in Prague, 2013

[11] McKenney P. E., Slingwine J. D.: *Read-Copy Update: Using Execution History to Solve Concurrency Problems*, in the Proceedings of Parallel and Distributed Computing and Systems, ACTA Press, 1998

[12] The HelenOS Team: *HelenOS 0.2.0 Design Documentation* (as of September 1st 2015), http://www.helenos.org/doc/design.pdf

[13] *HelenOS documentation* (as of September 1st 2015), http://www.helenos.org/documentation

[14] Kalibera T., Procházka M., Pizlo F., Děcký M., Vitek J., Zulianello M.: *Real-Time Java in Space: Potential Benefits and Open Challenges*, in the Proceedings of Data Systems in Aerospace (DASIA 2009), European Space Agency, 2009

[15] Lipavský L.: *Linux kernel userspace modules*, Master Thesis, Charles University in Prague, 2006

[16] Triplett J., McKenney P. E., Walpole J.: *Resizable, scalable, concurrent hash tables via relativistic programming*, in the Proceedings of the 2011 USENIX Annual Technical Conference, ACM, 2011

[17] *Microkernel* wiki entry at tunes.org (as of May 31st 2015), http://tunes.org/wiki/microkernel.html

[18] *Ticket #541: Hard real-time features* (as of September 1st 2015), http://trac.helenos.org/ticket/541