HelenOS project

project documentation

Contents

Chapter 1

Introduction

The HelenOS project[1] is an effort to develop portable and general purpose operating system. Operating systems in general are very non-trivial pieces of software. It takes many people, many months and many tools to design and develop even medium size and feature-limited kernel and userspace layer.

This report aims to document the development process of the HelenOS operating system as it is specified in [2] and as it has been carried out by the original six developers (i.e. Jakub Jermář, Ondřej Palkovský, Martin Děcký, Jakub Váňa, Josef Čejka and Sergey Bondari) in their work on Software project[1] at Faculty of Mathematics and Physics at Charles University in Prague. Other aspects of the wider HelenOS project (e.g. master theses related to the topic) are not discussed here.

1.1 How to read this document

Chapter 2 provides an insight into project's timeline, planning, development. It also presents some statistic data related to the HelenOS project.

Chapter 3 evaluates contributions and project dedication of each individual developer.

Chapter 4 gives thorough coverage of the third-party software involved with HelenOS and also experience comming from using that software.

[1] Software project is the name of a subject at MFF UK. It is supposed to last two semesters at least.

Chapter 2

Project

Chapter 3

Developers

3.1 Jakub Jermář

Jakub Jermář is the original author of the SPARTAN kernel and the initiator of the idea to start the HelenOS project. During the works on the system, he has been the most active developer. He also took on the project agenda and administration and became the project leader.

Before the project officially started, Jakub Jermář created the SPARTAN kernel for ia32 and mips32 along with SMP support, scheduler and synchronization[1].

In the project time proper, he implemented considerable parts of ia64 code and worked also on mips32 memory management. On the generic front, Jakub Jermář designed generic the virtual address translation interface for the 4-level hierarchical page table translation mechanism as well as for the global page hash table translation mechanism. He has been involved in address space management functions and userspace synchronization through futexes. Other areas of his contribution include the kernel console and the kernel ELF loader. Jakub Jermář is also the author of the generic buddy system framework and B+tree implementation.

3.2 Ondřej Palkovský

Ondřej Palkovský has been very agile HelenOS developer. He is responsible for large areas of the kernel and userspace and has completely created the amd64 port and completed the mips32 port to the extent that it runs on a real MIPS machine[2]. Besides the architecture specific involvement, Ondřej Palkovský programmed the slab allocator and modified the frame allocator to be self-contained which in turn let the old and very limited heap manager be removed from the kernel entirely. He also created architecture independent FPU lazy switching framework. Other

[1] The initial SPARTAN kernel did not reach userspace.

[2] SGI Indy

example of his activity is the IPC subsystem and partial TLS[3] support. Lastly, Ondřej Palkovský equipped the kernel console with features found in userspace command shells (e.g. tab completion of commands and command history through keyboard arrows) and wrote the kernel configuration software.

Apart from the self development, other members of the team appreciated Ondřej Palkovský's excellent debugging and troubleshooting skills. He has been the person behind discovery of majority of the compiler and simulator bugs.

3.3 Martin Děcký

Martin Děcký is a very dedicated HelenOS developer. Right from the beginning, Martin has cared about project's code purity and readibility. He was the first developer to start writing Doxygen-style comments. He has promoted the proper use of C language `const` keywords and extensive typedefing. On the tools front, he has rewritten the initial build system and created all our toolchain build scripts.

Martin worked and much improved the ia32 and amd64 kernel booting using the Grub bootloader and Multiboot specification. He also created specialized boot loaders for mips32 and ppc32 — architectures that don't provide many other ways to load userspace init tasks. Finally, Martin Děcký bacame the sole author of the entire ppc32 port and has encountered partial success in booting ppc64 port to real hardware[4].

3.4 Jakub Váňa

Jakub Váňa has worked on ia32 and ia64 FPU context switching and passive ia32 and active and passive ia64 console. He has relocated the ia64 kernel to region 7 (i.e. to the highest addresses) and has first coped with ia64 interrupts. Jakub Váňa has been always prepared to discuss different kernel issues. His expertise in ia32 is well recognized.

3.5 Josef Čejka

Josef Čejka has worked on ia32 memory map detection, softfloat and softint libraries and printf() standards comformance. He also ported several kernel libraries to userspace.

[3] Thread local storage.
[4] Apple G5.

3.6 Sergey Bondari

Sergey Bondari implemented sorting library functions and implemented the buddy allocator interface for the frame allocator. He edited project documentation.

Chapter 4

Software

During the development of the HelenOS operating system, we came across several types of software tools, programs, utilities and libraries. Some of the tools were used to develop the system itself while other tools were used to faciliate the development process. In some cases, we had a chance to try out several versions of the same product. Sometimes the new versions contained fixes for bugs we had discovered in previous versions thereof.

Another group of software we have used has been integrated into HelenOS to fill gaps after functionality that the genuine HelenOS code did not provide itself.

There is simply too much third party software that is somehow related to HelenOS to be covered all. This chapter attempts to present our experience with the key softare tools, programs and libraries.

4.1 Communication tools

Although the developers know each other in person, the development, with the exception of kernel camps, has been pretty much independent as far as locality and time goes. In order to work effectively, we have established several communication channels:

E-mail — We used this basic means of electronic communication for peer-to-peer discussion in cases when the other person could not have been reached on-line at the time his advice was needed or his attention was demanded. E-mail was also used for contacting developers of third party software that we needed to talk to.

Mailing list — As almost every open source project before us, also we opened mailing list for technical discussion. The advantage of having a mailing list is the fact that it enables multilateral discussions on several topics contemporarily, without the need for all the participants be on-line or even at one place. We have kept our first development mailing list closed to

public so that it seemed natural to us to use Czech as our communication language on the list since Czech, with one exception, is our native language and all of us speak it very well. Besides all the advantages, there are also disadvantages. First, communication over mailing list tends to be rather slow, compared for instance to ICQ. Second, because of its implicit collective nature, it sometimes tends to be so slow that an answer for a given question never comes.

Apart from the internal development mailing list, we have also used another mailing list for commit log messages which proved handy in keeping developers informed about all changes in the repository.

Finally, we have also established a public mailing list for communication about general HelenOS topics in English.

ICQ — Because we divided the whole project into smaller subprojects on which only the maximum of two people out of six would work together, the need for communication among all six people was significantly smaller than the need to communicate between the two developers who tightly cooperated on a specific task. For this reason, we made the biggest use of ICQ.

4.2 Concurrent versions systems

At the very beginning, when the SPARTAN kernel was being developed solely by Jakub Jermář, there was not much sence in using any software for management of concurrent versions. However, when the number of developers increased to six, we immediately started to think of available solutions.

We have begun with CVS because it is probably the best known file concurrent versions system. We have even had repository of HelenOS using CVS for a short time, but when we learned about its weaknesses we sought another solution. There are two weaknesses that have prevented us from using CVS:

- it is merely a file concurrent versions system (i.e. CVS is good at managing versions of each separate file in the repository but has no clue about the project's directory tree as a whole; specifically renaming of a file while preserving its revision history is next to impossible),
- it lacks atomic commits (i.e. should your commit conflict with another recent commit of another developer, CVS would not abort the whole operation but render the repository inconsistent instead).

Being aware of these limitations, we decided to go with Subversion. Subversion is, simply put, a redesigned CVS with all the limitations fixed. We were already familiar with CVS so the switch to Subversion was pretty seamless.

As for Subversion itself, it has worked for us well and has met all our expectations. Despite all its pros, there was a serious problem that occurred sometime in the middle of the development process. Because of some locking issues related to the default database backend (i.e. `Berkeley DB`), our Subversion repository put itself in a peculiar state in which it became effectivelly inaccessible by any means of standard usage or administration. To mitigate this problem, we had to manually delete orphaned file locks and switch to backend called `fsfs` which doesn't suffer this problem.

Other than that, we are happy users of Subversion. The ability to switch the entire working copy to particular revision is a great feature for debugging. Once we tracked a bug three months into the past by moving through revisions until we found the change that caused the bug.

4.3 Web tools

On our project website[1], we provided links to different web utilities that either functioned to access our Subversion repository or mailing list or provided another services:

Chora is a part of the Horde framework and can be used to comfortably browse Subversion repository from the web. We altered it a little bit to also show number of commits per developer on our homepage.

WHUPS is another component of the Horde framework. It provides feature request and bug tracking features. However, in the light of being rather closed group of people, we used this tool only seldomly. On the other hand, any possible beta tester of our operating system has had a chance to submit bug reports.

Mailman is a web interface to the mailing list we utilized. It allows to control subsriptions and search mailing list archives on-line.

4.4 Third party components of HelenOS

HelenOS itself contains third party software. In the first place, amd64 and ia32 architectures make use of GNU Grub boot loader. This software replaced the original limited boot loader after the Kernel Camp 2005 when Martin Děcký had made HelenOS Multiboot specification compliant. Because of Grub, HelenOS can be booted from several types of devices. More importantly, we use Grub to load HelenOS userspace modules as well.

Another third-party piece of the HelenOS operating system is the userspace `malloc()`. Rather than porting our kernel slab allocator to userspace, we have

chosen Doug Lea's public domain `dlmalloc` instead. This allocator could be easily integrated into our uspace tree and has proven itself in other projects as well. Its derivative, `ptmalloc`, has been part of the GNU C library for some time. However, the version we are using is not optimized for SMP and multithreading. We plan to eventually replace it with another allocator.

4.5 Build tools

Assembler, linker and compiler are by all means the very focal point of attention of all operating system projects. Quality of these tools influences operating system performance and, what is more important, stability. HelenOS has been tailored to build with GNU `binutils` (i.e. the assembler and linker) and GNU `gcc` (i.e. the compiler). There is only little chance that it could be compiled and linked using some other tools unless those tools are compatible with the GNU build tools.

As our project declares support for five different processor architectures, we needed to have five different flavors of the build utilities installed. Interestingly, flavors of `binutils` and `gcc` for particular architecture are not equal from the point of view of cross-binutils and cross-compiler installation. All platforms except ia64 require only the `binutils` package and the `gcc` package for the cross-tool to be built. On the other hand, ia64 requires also some excerpts from the ia64-specific part of `glibc`.

Formerly, the project could be compiled with almost any version of `binutils` starting with 2.15 and `gcc` starting with 2.95, but especially after we added partial thread local storage support into our userspace layer, some architectures (e.g. mips32) will not compile even with `gcc` 4.0.1 and demand `gcc` 4.1.0. Curiously, ia64 will not link when compiled with `gcc` 4.1.0.

As for the mips32 cross-compiler, Ondřej Palkovský discovered a bug in `gcc` (ticket #23824) which caused `gcc` to incorrectly generate unaligned data access instructions (i.e. `lwl`, `lwr`, `swl` and `swr`).

As for the mips32 cross-binutils[1], we observed that undefined symbols are not reported when we don't link using the standard target. We are still not sure whether this was a bug — `binutils` developers just told us to use the standard target and then use `objcopy` to convert the ELF binary into requested output format.

4.6 Virtual environments

After the build tools, simulators, emulators and virtualizers were the second focal point in our project. These invaluable programs really sped the code-compile-

[1] It remains uninvestigated whether this problem also shows with other cross-tools.

test cycle. In some cases, they were, and still are, the only option to actually run HelenOS on certain processor architectures, because real hardware was not available to us. Using virtual environment for developing our system provided us with deterministic environment on wich it is much easier to do troubleshooting. Moreover, part of the simulators featured integrated debugging facilities. Without them, a lot of bugs would remain unresolved or even go unnoticed.

From one point of view, we have tested our system on eight different virtual environments:

- Bochs,
- GXemul,
- msim,
- PearPC,
- QEMU,
- Simics,
- Ski,
- VMware.

From the second point of view, we have tested these programs by our operating system. Because of the scope and uniqueness of this testing and because we did find some issues, we want to dedicate some more space to what we have found.

4.6.1 Bochs

Bochs has been used to develop the SPARTAN kernel since its beginning in 2001. It is capable of emulating ia32 machine and for some time also amd64. Bochs is an emulator and thus the slowest from virtual environments capable of simulating the same cathegory of hardware. On the other hand, it is extremely portable, compared to much faster virtualizers and emulators using dynamic translation of instructions. Lately, there have been some plans to develop or port dynamic translation to Bochs brewing in its developer community.

The biggest virtue of Bochs is that it has traditionally supported SMP. For some time, Bochs has been our only environment on wich we could develop and test SMP code. Unfortunatelly, the quality of SMP support in Bochs was different from version to version. Because of SMP breakage in Bochs, we had to avoid some versions thereof. So far, Bochs versions 2.2.1 and 2.2.6 have been best in this regard.

Our project has not only used Bochs. We also helped to identify some SMP related problems and Ondřej Palkovský from our team has discovered and also fixed a bug in FXSAVE and FXRSTOR emulation (patch #1282033).

Bochs has some debugging facilities but those have been very impractical and broken in SMP mode.

4.6.2 GXemul

GXemul is an emulator of several processor architectures. Nevertheless, we have used it only for mips32 emulation in both little-endian and big-endian modes. It seems to be pretty featurefull and evolving but we don't use all its functionality. GXemul is very user friendly and has debugging features. It is more realistic than msim. However, our newly introduced TLS support triggered a bug in the `rdhwr` instruction emulation while msim functioned as expected. Fortunatelly, the author of GXemul is very cooperative and has fixed the problem for future versions as well as provided a quick hack for the old version.

4.6.3 msim

msim has been our first mips32 simulator. It simulates 32-bit side of R4000 processor. Its simulated environment is not very realistic, but the processor simulation is good enough for operating system development. In this regard, the simulator is comparable to HP's ia64 simulator Ski. Another similar aspect of these two is relatively strong debugger.

Msim has been developed on the same alma mater as our own project. All members of our team know this program from operating system courses. Curiously, this simulator contained the biggest number of defects and inaccuracies that we have ever discovered in a simulator. Fortunately, all of them have been eventually fixed.

4.6.4 PearPC

PearPC is the only emulator on wich we have run ppc32 port of HelenOS. It has no debugging features, but fortunatelly its sources are available under an open source license. This enabled Ondřej Palkovský and Martin Děcký to alter its sources in a way that this modified version allowed some basic debugging.

4.6.5 QEMU

QEMU emulates several processor architectures. We have used it to emulate ia32 and amd64. It can simulate SMP, but contrary to Bochs, it uses dynamic translation of emulated instructions and performs much better because of that.

4.6.6 Simics

4.6.7 Ski

4.6.8 VMware

4.7 Authoring tools

Bibliography

[1] HelenOS project, http://www.helenos.eu.

[2] HelenOS specifications