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

The HelenOS project was formed in late October 2004, when the six developers grouped and decided to adopt previous work of Jakub Jermář on the SPARTAN kernel[1] as a foundation for their new operating system.

2.1 Specification

The team had then worked on a specification[2] until March 8, 2005. The specification was based on Martin Děcký's draft and incorporated many suggestions from other members of the team. The biggest part of the discussion was concerned about how many and what processor architectures we will support. At that time, the SPARTAN kernel supported ia32 and mips32 to the extent that kernel threads could be scheduled. The ia32 port could do some very basic virtual memory operations and was capable of SMP service. Moreover, the mips32 port ran only in the msim simulator. None of them supported userspace threads.

We realized the need to support at least one 64-bit architecture and have long discussed whether it should be amd64 or ia64. We also considered ppc64. At the end, we decided to declare support for three new architectures, five architectures in total. Both amd64 and ia64 made it to the specifications, as well as PowerPC. As for PowerPC, the specification didn't say whether ppc32 or ppc64 or both will be supported.[2]

It is worth noting that we wanted to be sure of access to respective hardware or at least simulator, prior to committing to support particular architecture. The decision to support almost all suggested architectures[3] came after we had known for sure the above condition was satisfied.

[1]The SPARTAN kernel has been developed by Jakub Jermář since 2001.

[2]This has later proven a bit problematic because it is not very clear what ppc32 should be (i.e. the 32-bit G4 processor is not compatible with the 32-bit mode of the G5 processor.

[3]Namely, we didn't declare support for sparc64, but it got supported anyway as part of Jakub Jermář's master thesis.

We constructed our specification so that it contained a well defined set of mandatory features of the kernel and the userspace layer that had to be implemented. Besides the mandatory features, there was also an optional part comprising of three research or experimental topics. We hoped to eventually find time to work on them.

2.2 Project meetings

After adopting our specification, we started to meet regularly every two weeks for the sake of consultations. The regular meetings were cancelled only during the exam periods and summer holiday. The first meeting took place on April 28, 2005. There were exactly twenty two project meetings before 0.2.0 release.

The Faculty of Mathematics and Physics officially opened our project on June 10, 2005. However, serious collective work on the project, preceeded by individual efforts of some team members, began two months later.

2.3 Planning work

In the beginning, we structured our work by creating three two-member teams, each dedicated to one new architecture (i.e. amd64, ia64 and ppc32). However, dividing into couples didn't work out for the amd64 and ppc32 teams. In the end, both of those architectures were supported only with one member of respective team. This might have been because of two factors. First, the collective responsibility for the project allowed the less motivated members to work less than others. Second, over the time, some developers profiled out to be good at specific tasks to which they later adhered and were forwarded more similar work. It was generally accepted within the team if one of the couple traded one architecure-specific task for another task on HelenOS.

2.4 Kernel camps

There were two really important moments in our development process. Both of them took place in Harrachov, Czech Republic, where five team members moved two times, each time for a week of full-time intensive HelenOS development. These actions were called Kernel Camp 2005 and Winter Camp 2006. The former camp took place in August 2005 and was focused on getting all the architectures into our source tree and deepening their support. The latter camp took place in March 2006 and was dedicated to userspace support. In fact, we made the second camp the deadline for userspace milestone. With the exception of ppc32, all ports had some support for userspace prior to the second camp. Both of the camps moved the project miles ahead.

2.5 Coding style

We have adopted common coding style specification in order to improve code readibility and maintainability. Even though the specification relates only to stylistic matters, following it has the potential to encourage and improve cooperation within the team and provide good preconditions for future project growth.

Chapter 3

Developers

3.1 Jakub Jermář

Jakub is the original author of the SPARTAN kernel and the initiator of the idea to start the HelenOS project. Once the team was composed, he implemented considerable parts of the ia64 code and he also worked on the mips32 memory management. On the generic front, Jakub designed the generic virtual address translation interface for the 4-level hierarchical page table translation mechanism as well as the global page hash table translation mechanism. He has been involved in the address space management functions and userspace synchronization through futexes. Other areas in which he contributed include the kernel console and the kernel ELF loader. Jakub is also the author of the generic buddy system framework as well as the B+tree implementation. His latest contribution is the userspace PCI driver.

3.2 Ondřej Palkovský

Ondřej has completely created the amd64 port and completed the mips32 port to the extent that it runs on a real MIPS machine[1]. Besides the architecture specific involvement, Ondřej 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 example of his activity is the IPC subsystem and partial TLS[2] support. Lastly, Ondřej 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.

[1]SGI Indy

[2]Thread local storage.

3.3 Martin Děcký

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 bacame the sole author of the entire ppc32 port and has encountered partial success in booting ppc64 port to real hardware[3].

3.4 Jakub Váňa

Jakub has worked on ia32 and the 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. Lastly, Jakub programmed the VESA frame buffer support for ia32 and amd64 and created the userspace framebuffer driver.

3.5 Josef Čejka

Josef has worked on ia32 memory map detection, softfloat and softint libraries and printf() standards conformance. He also ported several kernel libraries to userspace and implemented considerable parts of HelenOS libc. Josef is the author of the userspace keyboard driver.

3.6 Sergey Bondari

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

[3]Apple G5.

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 software 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 the 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.

Next, the `pci` userspace task is using the `libpci` library. The library was simplified and ported to HelenOS. Even though filesystem calls were removed from the library, it still heavily depends on `libc`. By porting `libpci` to HelenOS, we demonstrated that applications and libraries are, given enough effort, portable to HelenOS.

Finally, we demonstrated the idea presented in the previous paragraph by porting over 13 years old BSD game of `tetris` to HelenOS. This particular version of tetris looks almost the same both on other people's operating systems and on HelenOS. Similar to `libpci`, `tetris` had to be modified in order to compile and run. The filesystem calls were removed or replaced as well as references to terminal I/O calls.

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`[3] (i.e. the assembler and linker) and GNU `gcc`[4] (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 or newer.

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-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 which 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.

Using several virtual environments for testing one architecture is well justified by the fact that sometimes HelenOS would run on two and crash on third or vice versa. Sometimes we found that it runs on real hardware but fails in a simulator. The opposite case was, however, more common. Simply put, the more configurations, no matter whether real or virtual, the better.

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.

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

4.6.1 Bochs

Bochs[6] 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 which 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. Moreover, it is possible to use the GNU debugger `gbd` to connect to running simulation, but this has also proven not very useful as we often needed to debug problems that existed only in multiprocessor configurations, which `gdb` does not understand.

4.6.2 GXemul

GXemul[12] 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[8] 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[7] is the only emulator on which 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[13] 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.

This emulator seemed to realistically emulate the `hlt` instruction, which was nice for those of us who use notebooks as their development machine.

Similar to Bochs, QEMU simulation can be aided by `gdb`. Debugging with `gdb` can be pretty comfortable[2] until one needs to debug a SMP kernel running on multiple processors.

4.6.6 Simics

Virtutech's Simics[10] simulator can be compared to a Swiss-army knife for operating system debugging. This proprietary piece of software was available to us under an academic license for free.

Simics can be set to simulate many different configurations of many different machines. It has the most advanced debugging features we have ever seen. To highlight some, its memory access tracing ability has been really helpfull to us. During device driver development, we appreciated the possibility to turn logging of the devices to a specified verbosity.

We used it to test and develop amd64 and ia32 architectures in SMP mode and mips32 architecture in UP mode. Simics emulates the 4Kc processor on the MIPS architecture. Unfortunately, this processor does not have an exception Reserved Instruction, which makes it unusable in an environment with programs using thread local storage.

Regardless of its invaluable qualities, it has still contained bugs. One of the most serious was bug with ticket #3351. Ondřej Palkovský discovered that its

[2]Especially when the kernel is compiled with `-g3`.

BIOS rewrites kernel memory during application processors start. Another bugs found were related to amd64 and mips32. As for amd64, Simics did not report general protection fault when `EFER.NXE` was 0 and a non-executable page was found (#4214). As for mips32, Simics misemulated `MSUB` and `MSUBU` instructions.

4.6.7 Ski

The ia64 port of HelenOS has been developed and debugged on the HP's IA-64 Ski[9] simulator. Ski is just an Itanium processor simulator and as such does not simulate a real machine. In fact, there is no firmware and no configuration tables (e.g. memory map) present in Ski! On the other hand, the missing parts can be supplied externally[3]. The simulator provides means of interaction with host system devices via Simulator SystemCalls (SSC). The simulator itself has graphical interface with pretty powerful, but not as good as those of Simics, debugging facilities.

Ski is a proprietary program with no source code available. Its binaries are available for free under a non-free license. It comes packaged with insufficient documentation which makes the development pretty problematic. For instance, there is no public documentation of all the SSC's. All one can do is to look into Linux/ia64-Ski port, which was written by the same people as Ski, and use it as a refernce. We had to look into Linux once more when our kernel started to fail in some memory-intensive stress tests. In fact, the problem was that the tests hit the IA-32 legacy videoram area. We fixed the problem, in the light of absence of any memory map, by blacklisting this piece of memory to our frame allocator.

The way HelenOS is booted on Ski is by simply loading its ELF image and jumping to it. The ELF header contains two fields describing where and how to load the program image into memory: VMA and LMA. VMA[4] is an address where the program's segment gets mapped in virtual memory. LMA[5] is the physical address where the segment is loaded in memory. Jakub Váňa discovered that Ski confuses VMA and LMA. This, what we believe to be a bug in Ski, has not shown in Linux since Linux always has LMA equal to VMA. People from the Ski mailing list had tried to help us but our repeated problem report didn't make it far enough for the HP to fix or at least clarify the issue. Finally, we adopted a workaround implemented by Jakub Jermář that simply swaps LMA and the program entry point in the kernel ELF image.

4.6.8 VMware

VMware[11] is the only virtualizer we have used in HelenOS development. It virtualizes the ia32 host machine. Since VMware version 5.5, we made use of

[3]This is actually how Linux runs in this simulator.

[4]Virtual Memory Address

[5]Load Memory Address

its possibility to run the guest system (i.e. HelenOS) on multiple processors. VMware has no support for debugging but is very useful for compatibility and regression testing because it's closest to the real hardware. VMware, being a virtualizer, is also the fastest of all the virtual environments we have utilized.

Appendix A

Goals and Achievements

A.1 Overall Conception

General-purpose and portable operating system with elements of microkernel design and fully preemptive kernel.

SPARTAN kernel created by Jakub Jermar will be used as a basis for further kernel development.

Detailed description of the features:

- General-purpose: Ready to run standard (non real-time) server and workstation applications. Support for common programming abstractions (threads, synchronization, physical and virtual memory management).
- Portable: Except small platform-specific kernel parts the system will be implemented in higher programming languages to be portable to different hardware platforms (PCs and similar).
- Fully preemptive kernel: The basic scheduling element will be a thread (more threads eventually grouped into a task) and the task switching will be preemptive in both user-space and kernel-space. However no real-time scheduling will be attempted.
- Fine grained locking in kernel: The kernel will not contain anything such as "big kernel lock", all critical sections will be handled with small granularity locking.
- Elements of microkernel design: The code running in kernel-space will be limited to a much smaller size compared for example to the traditional Unix design. The kernel will contain mostly just the code which is necessary to run in kernel-space (scheduling, memory management and protection, hardware resource management, IPC). Device drivers, filesystems, network stacks, etc. will be implemented in user-space.

The overall conecption of the kernel design was completely met. The kernel is fully preemptible, SMP ready with fine-grained locking. If possible, device drivers are implemented as standalone userspace tasks. HelenOS fully supports statically linked tasks. Both userspace tasks and kernel tasks are supported (N:M multithreading model).

The kernel was successfully ported to 5 architectures with one other architecture to come. The interfaces in the kernel are designed in such a way to fully utilize specifics of every platform, e.g. ASID and RID allocation in MIPS and IA64, two stacks for IA64 and SMP routines.

A.2 Research Domains

Following features can be eventually implemented as research subjects, but are optional to the overall design of the system:

- Kernel-level virtualization: Apart from some standard security model (i.e. unix-like or any other) the OS might support kernel-level context separation allowing to run more virtual operating environments on a single physical machine.

  Kernel-level virtualization was not attempted, although the microkernel design by itself allows completely different namespace simply by connecting the task to different name service daemon. Because new IPC connections can be created only through existing paths in the graph of the connections, messages can never flow between unconnected components of the graph.

- Framework for running GNU/Linux applications: There should be no syscall or native API compatibility, but rather some kind of compile-time layer (libc and other shared libraries) allowing to compile common GNU/Linux applications from sources.

  Two applications were ported with little effort - libpci and tetris. The porting of the tetris consisted mainly in rewriting termios dependent code. The libc library contains emulation layer for the most common functions.

- Object/message paradigm: In the contrary to Unix file paradigm (where every object in the system is represented by a file - even if there is no consistent mapping from the given object's methods to generic file methods), HelenOS might have a tree of objects instead of a tree of files. Each object in the tree can support an arbitrary set of messages and files are those objects which support the set of messages representing file methods (i.e. open, close, read, write, seek, etc.). All objects might support several compulsory messages (GetName, GetSupportedMessages, etc.). The message passing mechanism will be synchronous.

Every IPC message contains a field that specifies method number. However, tree of objects or any more complex functionality were not implemented.

However, because we have decided to use asynchronous message passing, a framework was needed to facilitate reasonably synchronous application view. This framework, heavily using userspace thread switching, allows writing transparent applications without the hassle usually connected with asynchronous applications, at the same time being easily portable to kernel-threaded environment.

A.3 Particular features

- Kernel features
  - Preemptive multiprocessing, SMP support, threads (tasks)
    - Simple scheduler (but more complex than round-robin), with threads as basic scheduling element. *Achieved.*
    - Support for thread priorities (possibly classes of priorities for user-space tasks). *Achieved*
    - Support for SMP CPU bounding. *Achieved.*
    - Utilization of non-boot CPU(s). *Achieved.*
    - Support for user-space threads (tasks as sets of threads). *Achieved.*
    - Support for kernel threads (independent code executed within the kernel) *Achieved.*
  - Kernel synchronization primitives, small granularity synchronization (preemptive kernel)
    - Semaphores, mutexes, condition variables, RW-locks, spin-locks, etc. *Achieved.*
    - No "big kernel lock". *Achieved.*
  - Physical and virtual memory management
    - Proper handling of physical memory regions. *Achieved.*
    - Physical memory heap (allocating of continuous blocks of physical memory). *Achieved.*
    - Arbitrary number of independent virtual memory mappings (both for threads and internal kernel usage). *Achieved.*
    - Kernel allocator in virtual memory (buddy/slab). *Achieved.*
    - Named (text, stack, heap) and unnamed virtual memory areas. *Achieved.*
    - Copying and sharing pages between different memory mappings. *Achieved.*

- Basic hardware handling
    - Handling of basic boot-time hardware (CPU, PCI buses, memory, display, keyboard, RTC, etc.) in kernel. *Achieved.*
    - Handling of specific hardware resources which are fundamentaly unreachable from user-space on given platform. *Achieved.*
- IPC, user-space hardware access framework
    - Abstraction for implementing inter-process communication (message passing, etc.). *Achieved.*
    - Interface for enabling the user-space threads to gain access and manage hardware resources (with kernel modules where needed). *Achieved.*
- User-space features
    - Basic API
        - Memory management API (memory regions creation, descruction, resizing). *Achieved.*
        - Task/thread management API. *Achieved.*
        - Synchronization API. *Futexes implemented.*

A.4 Implementation details

- Supported platforms
    - Real hardware support
        - IA-32 (will be tested on multiple consumer Intel Pentium 4, Intel Pentium M, AMD Athlon XP and AMD Athlon MP machines) *Runs on comodity hardware. Tested on several multiprocessor computers.*
        - PowerPC (will be tested on a consumer IBM PowerPC G5 machine) *To some extent runs on the G4 machine. G5 machine is a 64-bit architecture completely different from 32-bit port that was attampted.*
    - Emulated support
        - MIPS (will be tested in MSIM R4000 simulator) *Tested in msim, gxemul and partially in simics simulators. Booted kernel on SGI Indy, however no real hardware input/output support was attempted.*
        - IA-64 (will be tested in Ski simulator) *Tested in Ski simulator.*
        - AMD64 (will be tested in Simics simulator) *Tested on single-processor computer. Runs in simics, bochs and qemu simulators.*

References

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

[2] HelenOS specifications

[3] binutils, http://www.gnu.org/software/binutils/

[4] gcc, http://gcc.gnu.org/

[5] GRUB, http://www.gnu.org/software/grub/

[6] Bochs, http://bochs.sourceforge.net/

[7] PearPC, http://pearpc.sourceforge.net/

[8] msim, http://nenya.ms.mff.cuni.cz/~holub/msim

[9] Ski, http://www.hpl.hp.com/research/linux/ski/

[10] Simics, http://www.virtutech.com/about/research

[11] VMware, http://www.vmware.com/products/desktop/ws_features.html

[12] GXemul, http://gavare.se/gxemul/

[13] QEMU, http://fabrice.bellard.free.fr/qemu/