VIREOS: An Integrated, Bottom-Up, Educational Operating Systems Project with FPGA Support

Marc L. Corliss
Hobart and William Smith Colleges
Dept. of Mathematics and Computer Science
Geneva, NY
corliss@hws.edu

Marcela Melara
Hobart and William Smith Colleges
Dept. of Mathematics and Computer Science
Geneva, NY
marcela.melara@hws.edu

ABSTRACT

In this paper, we present the VIREOS project, a new operating system designed specifically for the classroom. VIREOS is a simple, Unix-like, operating system, which runs on the Larc educational architecture. A VIREOS/Larc system can either be simulated or run on a pre-configured FPGA. The VIREOS project is well integrated with an introductory computer architecture course using Larc and the assignments are structured in a similar fashion: using a bottom-up approach. We have several resources available on the Web, which help reduce the overhead of adopting VIREOS. Finally, VIREOS has been used in one operating systems course already, and the feedback from students was generally favorable.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design; K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Design

Keywords

VIREOS, operating systems, education

1. INTRODUCTION

The operating system (OS) is a core component in modern computers and naturally an important area of study for undergraduate computer science students. Obviously, operating systems are ubiquitous in modern computers, and thus, students should know something about them. Moreover, perhaps more than any other course in computer science, an operating systems course ties together all aspects of computer system design. After all, it is the responsibility of the OS to manage application software running on the computer as well as to provide that software with an abstract view of the hardware.

SIGCSE'11, March 9–12, 2011, Dallas, Texas, USA.

Naturally, the best way for a student to learn how the operating system works is to build one. Of course, a student cannot be expected to build a commercial-caliber OS such as Linux from scratch. The student must be either given a substantial amount of starting code or the OS must be simple enough that it can be completed in a single semester. In this paper, we explore the latter approach; we introduce and describe a new operating system designed specifically for the classroom.

The operating system, which we call VIREOS (a Vanilla, Introductory, but Realistic, Educational Operating System), is a simple, Unix-like OS written entirely in C. VIREOS is implemented on top of the Larc classroom architecture [5], which has a much simpler instruction set architecture (ISA), input/output (I/O) interface, and memory management hardware support (among other things) than most commercial architectures. VIREOS was used in an operating systems course in fall 2009 at Hobart and William Smith Colleges, and overall, it was well received by the students.

VIREOS has several virtues that make it well suited for the classroom. First, although it was designed for simplicity, VIREOS includes many important features of modern operating systems such as a Unix-like file system (using i-nodes) and virtual memory via paging. Second, though a VIREOS/Larc system is often simulated, we also provide support for running it on a pre-configured FPGA. Third, VIREOS can be integrated with a course on computer architecture (especially one using Larc). The VIREOS assignments are structured in a bottom-up fashion so that students are always implementing new components on top of components they have already built and understand. Finally, we have several resources including comprehensive documentation and code available on the Web at *http://math.hws.edu/vireos*.

In the remainder of this paper, we describe the VIREOS project in more detail. In Section 2, we discuss related work. In Section 3, we enumerate our goals in designing the VIREOS project. In Section 4, we give an overview of the VIREOS project. In Section 5, we share our experiences using the VIREOS project in the classroom. Finally, in Section 6, we present future work.

2. RELATED WORK

There are many existing educational operating system projects (too many to cite them all), but there are three main approaches. One approach is to have students modify an existing, commercial operating system rather than write one from scratch [6, 9, 13]. Obviously, the advantage of

this approach is that the student works with a commercial OS. They can also potentially look more deeply into issues of performance and security. But the disadvantage is that the student misses out on implementing some aspects of the OS. As a result, the student may leave the class with an incomplete view of some parts of the OS.

A second approach is to have students build a complete OS for a commercial machine [2, 3, 8, 11, 12, 18]. The OS, itself, may also have commercial uses, which is the case for Minix [18]. For Minix though, we believe some students may find the software infrastructure daunting. Even with a simpler infrastructure, using a commercial machine, while certainly appealing, has some drawbacks. In particular, many aspects of a commercial machine, such as interrupt handling, I/O, and memory management, are complicated and can make the job of the student more challenging. Some of this complexity can be traded off for features in the operating system such as paging and more complex file systems. But, in this case, as with modifying an existing OS, the student may have gaps in their understanding of some parts of the OS.

A third approach, which is taken in the VIREOS project, is to build an operating system for a simpler, non-commercial machine; one that is designed specifically for the classroom. Nachos [4] and OS 161 [7] are two prominent examples that fall into this category. With this approach some complexity is naturally avoided (with regards to the underlying machine) at the expense of some loss of realism. We address this deficiency in the VIREOS project to some extent by configuring an FPGA to run as a VIREOS system. Although it is probably not practical for use in assignments, it can be used to demonstrate to the student that their operating system will be a part of a complete, working system. In fact, this use of hardware in the VIREOS project is similar in many educational OS projects, even those running on commercial hardware. During OS implementation and testing, emulation or simulation is often used.

One aspect of VIREOS that is different from many educational operating systems projects is the potential for integration with other computer science courses. In particular, the machine that VIREOS runs on top of, Larc [5], was designed specifically for an introductory computer architecture class (sometimes called computer organization). The approach in both projects works similarly: each new assignment builds on top of previous work done by the student. Therefore, operating systems instructors can leverage student's knowledge and experience gained in the architecture course. In the future, we plan on exploring connections with other courses such as an advanced computer architecture course.

Embedded XINU [3] is a recent project similar to VIREOS in terms of its integration with other computer science courses. But Embedded XINU differs from VIREOS in that it targets embedded systems and runs on top of a Linksys wireless router. On the one hand, Embedded XINU has the virtue that the operating system runs on real hardware. But at the same time, this hardware must be installed and maintained by the instructor, which adds some overhead to its adoption. With VIREOS, the use of hardware (*i.e.*, FPGAs) is optional.

3. GOALS

We had several goals in mind when designing the VIREOS project. Here we list each goal along with a description of how it was met.

- **From scratch.** First, we want students to write the operating system basically from scratch (with just a little auxiliary starting code). Of course, the OS and/or machine must be simple for the student to complete the OS in a single semester. To achieve this goal, we use an education-based architecture (Larc [5]). We also primarily do not use any interrupts (except in a single case). We have found that these two simplifications are enough to ensure that the OS is doable in a single semester without sacrificing some other important and interesting aspects of the OS (*e.g.*, file system functionality and memory management).

- **Good coverage.** Although the machine and OS are simplified in some ways, we still want the OS to contain as many features as possible. For instance, VIREOS has a Unix-like file system (using i-nodes), multi-processing, and virtual memory. We have found from experience in teaching operating systems that some students have trouble fully understanding these aspects of the OS unless they have actually implemented them.

- **Realistic.** Even though we use an educational architecture, we still want students to feel that they are building a "realistic" operating system. We have configured an FPGA to run the VIREOS system so that instructors can demonstrate VIREOS running on real hardware. In addition, we modeled parts of VIREOS after *nix operating systems, which students at our institution use when doing their computer science work. In particular, each VIREOS system call is similar to a corresponding call in Unix/Linux and the file system is implemented using i-nodes.

- **Well integrated.** One very important goal for us was integrating the operating systems course with our introductory course in computer architecture. For the student, the study of operating systems is a natural extension from their study of computer architecture. Consequently, we implemented VIREOS to run on the Larc classroom architecture [5], which we use in our architecture course. It should be pointed out, however, that VIREOS can be used at institutions where Larc is not adopted in the architecture course.

- **Bottom-up.** Another important goal for us was to use a bottom-up approach in structuring the assignments. Each assignment should build off of the previous assignment; the student should never work with code that they do not yet understand. In the first VIREOS project assignment (minus a preliminary assignment where the student builds a shell), the student builds a basic trap and I/O handler, and then in later assignments adds additional functionality (a file system, process manager, and memory manager).

- **Well documented and supported.** Finally, we want to provide thorough documentation and support

to lower overhead for instructors to adopt the project and for students to use it. We have written a lab manual, which includes in-depth descriptions of both the OS and the hardware, as well as a detailed write-up for each assignment. The manual is freely available off the Web at *http://math.hws.edu/vireos* along with the API for all the assignments. The solution code can also be requested by instructors (and instructors only – see the VIREOS web page for details). The instructor can also obtain Verilog files for configuring an FPGA (the Altera DE1 [1]) to run the VIREOS/Larc system.

4. VIREOS PROJECT

This section describes the VIREOS project in more detail. Section 4.1 describes the VIREOS system and Section 4.2 describes the assignments and the available resources for both students and instructors.

4.1 VIREOS System

General. VIREOS is a *nix-like (Unix/Linux) operating system (although greatly simplified). VIREOS has support for a file system, time sharing, and virtual memory via paging. The file system support includes both files and directories although it does not include links. Unlike a *nix system, however, VIREOS does not have support for multiple users. It also mostly does not provide security (*e.g.*, the user can read/write any file) although it does include some safety measures to prevent an errant program from corrupting the system.

The VIREOS system calls enable user programs to work with files and directories, as well as manage processes. Each of these system calls is similar to a corresponding *nix system call. VIREOS also has support for communicating directly with I/O devices such as the disk. Although this is not ideal from a safety/security standpoint (*e.g.*, an errant application could corrupt the disk), it makes it much easier for students to test their operating system implementation. (Note: these system calls could be disabled in a working version of VIREOS.)

Source language. VIREOS is written entirely in a C-like language called C--, which was first developed by Jim Lenz [10], although it has been significantly extended for the VIREOS project. In the future we plan on implementing VIREOS in full ANSI C, but in order to get the project up-and-running faster, we opted to initially use a simple, easily extendable compiler. Moreover, the C-- compiler adds some OS-specific features such as providing an array for saving and restoring registers at the source language level. Because of these features, the OS can be written entirely in C-- (no extra assembly code is required).

C-- is missing some features from C including multi-dimensional arrays, unions, nested functions/structs, typedefs, switch statements, do/while loops, function pointers, and primitives besides int and char (among other things). But C-- retains most of the features of C and in practice the programming experience is much the same.

Target machine. VIREOS runs on the Larc architecture [5]. Larc is a simple, 16-bit, MIPS-like architecture designed specifically for the classroom. It can support most text-based (non-graphical) systems such as those that use a command-line interface (*i.e.*, a shell) for entering commands and running programs. A Larc machine is primarily simulated although we have configured a Larc processor on an FPGA (discussed below).

Larc has ISA support for four I/O devices: a keyboard, a text-based monitor, a disk (the block size is 1/2 KB), and a real-time clock. To simplify the OS, polling is used to communicate with I/O devices. The only supported interrupt is a timer for implementing time sharing. In the future, we will explore adding support for interrupt-driven I/O.

Each I/O device includes a set of registers for communication between the CPU and that device. These are programmed via memory-mapped I/O. There are also a set of memory-mapped registers for managing the CPU.

Larc has support for two memory management techniques: virtual memory via paging as well as a simple sandboxing technique (using a base and limit register) [17].

Run environment. The VIREOS/Larc system can be run in one of two ways. First, it can be run in a Larc simulator, which is written in C and compiled to run on most Linux machines. It can also be run on a pre-configured Altera DE1 FPGA [1] (other FPGAs could potentially be used by modifying our provided configuration files). When students are implementing VIREOS, it probably makes more sense for them to use the simulator as there is no hardware overhead. But the FPGA can be used to demonstrate a "real" VIREOS/Larc system. There is also the potential for integrating this project with a course on microarchitecture.

Note: we are currently in the process of adding paging support to the FPGA but we expect to have this completed by the time of publication. All other OS features are supported, and as a result, the FPGA can still be used for demonstration purposes as well as for the non-paging course assignments.

Code organization. VIREOS consists of five components: a trap handler, an I/O handler, a file system manager, a process manager, and a memory manager. Each component is contained in a separate C module. Students implement each of these components over the course of the semester.

In addition to these components, there are also a couple of modules for performing auxiliary functionality such as manipulating strings or converting between data types. These modules are provided to students.

4.2 Assignments and Resources

Assignments. Table 1 lists each of the assignments students undertake in the VIREOS project. There are 5 total assignments. the first 2 are shorter two-week assignments and the last 3 are longer three-week assignments. Students start by implementing an interactive shell (a two-week assignment) with support for redirection, piping, and background processing. The shell serves as the core application in a VIREOS system and it is important that students understand how an application like the shell works. This assignment also gives students a better understanding of the OS functionality, particularly with regards to process management, which they may not have fully understood prior to completing the assignment.

In the remainder of the assignments students implement the various components of the VIREOS kernel. As much as possible, these assignments are ordered in a bottom-up fashion; students build each component on top of components built in previous assignments.

Assignment	Component	Length	Details
0	Interactive Shell	2 weeks	Supports redirection, background processing, and piping
1	Trap and I/O handler	2 weeks	Uses polling
2	File system manager	3 weeks	Uses i-nodes, supports files (for base credit) and directories (for extra credit)
3	Process manager	3 weeks	Uses round-robin scheduling, supports fork, exec, wait
4	Memory manager	3 weeks	Uses paging

Table 1: VIREOS project assignments.

At the start of each assignment, the student is given a precompiled, working version of the components implemented in earlier assignments. In some cases, these must be slightly augmented to work with the new component. However, these augmentations are generally small.

At the end of each assignment, the student has a working kernel although it may not include some functionality, which will be implemented in a later assignment. The student is given a set of test programs, which they can use to test their implementation of the new component(s). The student is also given a compiled, reference operating system to see how the test programs should behave with a working OS.

In the first kernel assignment, a two-week assignment, students build a trap and I/O handler. This kernel will support applications that directly communicate with I/O devices, several of which the student is provided with. Of course, this kernel will not support applications that make use of the file system or create processes.

In the second kernel assignment, a three-week assignment, students build a file system on top of their trap and I/O handler. For base credit, the student must support all the file-related (and not directory-related) system calls. The directory-related system calls are saved for extra credit. The student is given several test programs that manipulate files (and potentially directories) as well as stress test the file system. Of course, none of these test programs include multiprocessing.

In the third kernel assignment, a three-week assignment, students build a process manager with a simple round-robin scheduler. With the process manager, VIREOS has support for the fork, exec, and wait system calls (among others) and time sharing. However, the kernel is highly inefficient. Because paging is unimplemented and so each process can use the entire user space, the kernel saves and restores the entire address space on a context switch. In this assignment, the student is given a working shell along with several shell utilities (*e.g.*, **ls**, **mkdir**). The student is also given a program for stress testing the process manager.

In the final kernel assignment, a three-week assignment, students implement paging. Although this adds no additional functionality to the OS, it significantly improves its efficiency. Because the address space is small and the clock speed is slow, the entire page table can be stored in a translation lookaside buffer (TLB), which greatly simplifies the operating system. To test their implementation of paging, the student can use the same test programs from the process manager assignment along with one new test program for stress testing the memory manager.

Student/instructor resources. We have several, freely-available resources for both students and instructors using VIREOS. These can be attained off of the VIREOS web page at *http://math.hws.edu/vireos*. First, we have a comprehensive student manual (a PDF file), documenting all aspects of the VIREOS project. It includes informational sections on the Larc ISA, VIREOS code layout, and C-- language, as well as detailed write-ups for each assignment. Although this manual does not cover operating systems theory, it is intended to be used alongside a traditional operating systems textbook [14, 15, 17].

We also have the API for each assignment available on the web page. It is generated from comments in the source code using the tool doxygen [19], which works similarly to javadoc [16].

We also provide instructors and students with the incomplete toolset, which the student must complete over the course of the semester. Instructors (and instructors only) can get the solution code for the assignments. Instructors can also get the Larc simulator source code as well as the Verilog code used to configure the FPGA. Details are available on the web page for requesting this restricted code, although instructors should expect some delay as each request must be validated.

5. EXPERIENCES

The VIREOS project was used in the fall 2009 offering of Operating Systems at Hobart and William Smith Colleges (HWS). We had not yet configured an FPGA to run VIREOS, but all other aspects of the project were fully implemented and tested in the class.

The class was taught by the first author of this paper. The class consisted of a lecture (3 hours per week) and a lab (1 and 1/2 hours per week). The lab period gave students a chance to get help from the instructor on the programming assignments. The course was taught using the textbook by Tanenbaum [17], which was supplemented with the VIREOS documentation.

The size of the class was small with only 8 students in total. Interestingly, 3 of the 8 students were exchange students from a European university. The 5 HWS students had used Larc before while the 3 exchange students had not.

None of the students had any significant experience programming in C (which VIREOS is written in) although the 3 exchange students had some experience programming in C++ (the HWS students did not). As a result, the students were given a week-long primer on C in the beginning of the course along with a short, 1-week programming assignment designed to familiarize them with C. Students were also given a second 1-week assignment at the beginning of the course on using the interactive shell and its advanced features (*e.g.*, piping) prior to building their own. Note: these preliminary assigments would be lifted for students with more C and/or shell experience.

Overall, the VIREOS project was well received by the students in the class. At the end of each assignment, the students were asked whether they found the assignment challenging and interesting, whether they had a better under-

standing of how operating systems work, and whether the time allotted was reasonable (as well as a call for any other comments). All of the students in the class generally found each assignment challenging and interesting. The students also found that each assignment gave them a better understanding of how operating systems work. For example, one student commented on the file system assignment:

> This [assignment] was definitely challenging, but it really helped me get a grasp on the file system concepts we talked about in class. I found the translation from file system theory to implementation pretty interesting–it took a lot of work to make sure the underlying theory was working correctly.

A second student commented on the process manager assignment:

> I learned a lot about implementing a process manager and all the complications that make it challenging. Working on this project really did help me to understand how operating systems manage multiple processes.

There were other similar comments across the various assignments. In addition to this OS-specific feedback, several students also felt that the assignments made them much better C programmers (a relatively new language for them) as well as code testers. One student mentioned on the file system assignment:

> I also am beginning to feel pretty comfortable with C since so much of its ins and outs is used to program the file system.

At the end of the semester, we also asked the 5 HWS students, who had taken both the operating systems and introductory architecture courses at HWS, if they found the integration between the two courses helpful in their overall understanding of computer systems. All of the students agreed that it was beneficial using Larc in both courses. One student wrote:

> I think the integration of the courses was immensely helpful in giving me a better understanding of computer systems. The hands on design and coding experience was a very valuable part of learning the course material, and using Larc ... across the board made it easier to jump into each new project.

There were also some negative comments. Many students found errors or omissions in the write-up, which have subsequently been fixed. Some students commented that the assignments had a significant learning curve and it was hard to get started. We have modified the assignment write-ups with this in mind.

In addition, some students felt that it was hard to complete the assignments in the time allotted. However, often these same students commented that they started the assignment too late. On average, students reported working 7-9 hours outside of class (for all the course work), which we thought was reasonable. Furthermore, for each project, nearly all the students submitted working code albeit with some minor bugs.

Finally, many students commented that testing was difficult. Most of these comments were made on the process management assignment. One student states:

> This is a difficult project to get fully working because it can be very hard to debug given the nature of the assignment. At the same time, I think it is a valuable assignment because it does a very good job of reinforcing the course material and it is interesting to try and implement. The problem is that it is so hard to get every detail working properly.

In the future, we plan to explore ways to make the debugging process easier for students, either by adding better test programs or by adding new debugging tools to the infrastructure. Other students commented that testing was slow, particularly for the process management assignment (which probably added to the difficulty in testing). In this assignment, memory is not paged (paging is added in the next assignment), and so, the processor manager is inefficient. We are looking for ways to address this in the future such as requiring, for this assignment, all concurrently-running programs to reside in memory in order to avoid expensive disk operations. This requirement would limit the size of the test programs (since they would all need to simultaneously fit in memory) but would make for faster testing.

We hope to address these negative comments in a future version of VIREOS. In general though, we are happy with the initial feedback we received from students.

6. FUTURE WORK

We plan to explore several directions of future work. First, we want to look at incorporating interrupt-driven I/O into the project. As this will represent more complexity and work for the student, we will give instructors the option of choosing between interrupt-driven I/O and polling.

We also want to look at incorporating new I/O devices such as a graphical monitor and a mouse. The closer the VIREOS system is to a commercial machine, the more appealing it will be to the student. But as with interrupt-driven I/O, we plan to make these I/O devices optional, and give instructors more flexibility in tailoring the project to their own class.

We also want to incorporate some security into VIREOS and potentially move from a single-user system to a multi-user system. Students would implement mechanisms for authenticating users and for protecting system resources (*e.g.*, files), among other things.

We also plan on migrating VIREOS from C-- to ANSI C. Although the programming experience is similar for the two languages, there are some differences, which can make it difficult when programming in C--.

In addition, we plan on adding a set of tools to ease the difficult debugging process. For instance, we hope to add a graphical debugger to the toolset to allow students to step through their operating system code. We will also look for other tools and resources to make debugging VIREOS easier for the student.

Finally, we plan to add applications to the VIREOS/Larc system to make it more usable and appealing to the student.

It currently includes only a shell, some basic shell utilities, and few simple (text-based) games.

7. CONCLUSIONS

The VIREOS project is a new, simple, Unix-like, classroom operating system running on the Larc educational architecture. The VIREOS project uses a bottom-up structure for assignments and can be well-integrated with a course on introductory architecture, which also uses Larc. The VIREOS project has many resources available on the Web, including the project API, solution code, and configuration files for running VIREOS on an FPGA. In fall 2009, we used VIREOS in the classroom and the feedback from students was generally favorable.

8. ACKNOWLEDGMENTS

Professor John Vaughn at Hobart and William Smith Colleges gave valuable input and advice on this work. Marcela Melara was supported by a generous grant from the Office of the Provost at Hobart and William Smith Colleges. Finally, the anonymous reviewers provided helpful feedback.

9. REFERENCES

[1] Altera Coorporation. *DE1 Development and Education Board – User Manual*, 2006.

[2] M. D. Black. Build an operating system from scratch: a project for an introductory operating systems course. In *SIGCSE '09: Proc. of the 40th ACM tech. symp. on computer science education*, pages 448–452, 2009.

[3] D. Brylow. An experimental laboratory environment for teaching embedded operating systems. In *SIGCSE '08: Proc. of the 39th SIGCSE tech. symp. on computer science education*, pages 192–196, 2008.

[4] W. A. Christopher, S. J. Procter, and T. E. Anderson. The Nachos instructional operating system. Technical report, 1993.

[5] M. L. Corliss and R. Hendry. Larc: A little architecture for the classroom. *Journal of Computing Sciences in Small Colleges*, 24(6):15–20, 2009.

[6] R. Hess and P. Paulson. Linux kernel projects for an undergraduate operating systems course. In *SIGCSE '10: Proc. of the 41st ACM tech. symp. on computer science education*, pages 485–489, 2010.

[7] D. A. Holland, A. T. Lim, and M. I. Seltzer. A new instructional operating system. In *SIGCSE '02: Proc. of the 33rd SIGCSE tech. symp. on computer science education*, pages 111–115, 2002.

[8] D. Hovemeyer, J. K. Hollingsworth, and B. Bhattacharjee. Running on the bare metal with GeekOS. In *SIGCSE '04: Proc. of the 35th SIGCSE tech. symp. on computer science education*, pages 315–319, 2004.

[9] O. Laadan, J. Nieh, and N. Viennot. Teaching operating systems using virtual appliances and distributed version control. In *SIGCSE '10: Proc. of the 41st ACM tech. symp. on computer science education*, pages 480–484, 2010.

[10] J. Lenz. Wisc C-- compiler. URL: *http://pages.cs.wisc.edu/∼lenz/compiler.html*, 2003.

[11] H. Liu, X. Chen, and Y. Gong. BabyOS: a fresh start. In *SIGCSE '07: Proc. of the 38th SIGCSE tech. symp. on computer science education*, pages 566–570, 2007.

[12] B. Pfaff, A. Romano, and G. Back. The Pintos instructional operating system kernel. In *SIGCSE '09: Proc. of the 40th ACM tech. symp. on computer science education*, pages 453–457, 2009.

[13] A. Schmidt, A. Polze, and D. Probert. Teaching operating systems: windows kernel projects. In *SIGCSE '10: Proc. of the 41st ACM tech. symp. on computer science education*, pages 490–494, 2010.

[14] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., 8th edition, 2008.

[15] W. Stallings. *Computer Organization and Architecture: Designing for Performance*. Prentice Hall, 7th edition, 2005.

[16] Sun Microsystems, Inc. *javadoc - The Java API Documentation Generator*, 2002. URL: *http://java.sun.com/j2se/1.5.0/docs/tooldocs /solaris/javadoc.html*.

[17] A. S. Tanenbaum. *Modern Operating Systems*. Pearson Prentice Hall, 3rd edition, 2008.

[18] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems: Design and Implementation*. Pearson Prentice Hall, 3rd edition, 2006.

[19] D. van Heesch. *Doxygen Manual*, 2010. URL: *http://www.doxygen.org/manual.html*.