# Using The Nanvix Operating System in Undergraduate Operating System Courses

Pedro Henrique Penna, Henrique Cota de Freitas, João Caram, Márcio Castro, Jean-François Méhaut

# Using The Nanvix Operating System in Undergraduate Operating System Courses

Pedro Henrique Penna,
Henrique C. Freitas and João Caram
Department of Computer Science
Pontifical Catholic University of Minas Gerais
Belo Horizonte – Brazil

Márcio Castro
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis – Brazil

Jean-François Méhaut
University Grenoble Alpes,
Inria, CNRS, Grenoble INP, LIG
F-38000 Grenoble – France

*Abstract*—**Operating Systems (OSs) have an important position in the Computer Science curriculum. When students face this subject, they study core concepts, mechanisms and strategies that apply to several fields. To support practical lectures in an OSs course, instructors may adopt an OS on which students can work, exercising their knowledge and enhancing their practical skills. In this context, we present Nanvix, a new OS designed to address this use in undergraduate OSs courses. We introduce a flexible assignment-based teaching methodology for our OS, and we assess the effectiveness of this methodology by applying it in the OSs course of the Pontifical Catholic University of Minas Gerais. When using Nanvix, the average score of the students in the course increased in 11.2%, and the failure rate dropped 47.7%. Moreover, we observed that with Nanvix students got more motivated and interested in the OSs field.**

## I. Introduction

Operating Systems (OSs) play an important role in the software stack. They act as a hardware manager by multiplexing access to resources, ensuring data protection and integrity, and enforcing protocols and policies. Beyond that, OSs extend the functionalities of the underlying hardware, presenting the user a programming interface that is more pleasant to deal with. In summary, without OSs, programmers would have to worry about machine intricacies, ending up to be less productive [1].

Given the importance of OSs nowadays, it is simple to highlight some extra points that turn them into a significant subject to Computer Science (CS) curriculum. First, OSs help students to design, implement and hack complex computing systems. Also, they introduce core concepts, mechanisms and strategies which can be applied to several fields of CS.

Teaching methodologies in OSs can follow several approaches. One that is widely adopted and mixes theory with practice works as follows. The instructor divides the OSs course into two parts: theoretical and practical lectures. In the former, he/she introduces students to the main concepts and principles of OSs, usually using some supporting material, such as textbooks. In this first part, the goal is to provide students with a solid foundations in the field. In practical lectures on the other hand, the instructor assigns activities to students so that they can exercise their theoretical knowledge. Here, the instructor often alternates between user- and kernel-level activities, possibly using some supporting OS [2], [3], [4], [5], [6], [7].

With this kind of software, students have at their disposal an OS on which they can work, exercising their theoretical knowledge and enhancing their practical skills. Unfortunately, existing OSs lack in several points. Some are restricted to run in simulators, while others run on hardware, but are too large to offer a pleasant learning rate to students; some provide no teaching material neither grading suggestions, whereas others impose a strict teaching methodology that may not reflect the instructor needs; and finally, some embrace outdated features, while others support modern features, but ask students to implement most of them, which can be overwhelming depending on the course schedule.

In this context, the goal of this work is to present Nanvix, a new OS that we designed and implemented from scratch to tackle the aforementioned problems. Nanvix may be used by instructors to introduce students to core OS concepts, which are common across the general purpose, real-time and embedded systems. In addition, this work introduces a flexible teaching methodology that may be used with Nanvix. We present an evaluation of our methodology and OS, when we applied both in the OSs course of the CS Undergraduate Program at Pontifical Catholic University of Minas Gerais (PUC Minas). Our work differs from previous ones in several points. First, in contrast to related software, Nanvix:

i. is smaller, hence making kernel hacking easier;
ii. is fully featured, thus enabling instructors to better meet their teaching needs;
iii. runs in virtual machines and on real hardware, which further motivate students;
iv. is shipped with an extensive documentation, hence saving instructor's time; and
v. presents an internal structure that is similar to Unix and targets Posix specification, thus making students face a realistic environment.

Second, this work introduces a flexible assignment-based teaching methodology that may be used with Nanvix. This type of methodology enables instructors to better meet their course schedule and it is absent in related OSs. Finally, this work presents a quantitative evaluation of the use of our OS. To the best of our knowledge, such evaluation is hardly found in the literature, thus reinforcing the our main contributions.

The remainder of this work is organized as follows. In Section II, we discuss related work. In Section III, we present the internals of Nanvix. In Section IV, we introduce the teaching methodology that may be used with our OS. In Section V, we present a qualitative and quantitative evaluation of our Nanvix-based teaching methodology. In Section VI, we expose our conclusions and future perspectives of this work.

## II. Related Work

To support practical lectures in OSs courses, several OSs narrowed for this goal are available. In the sections that follow, we briefly discuss some of the most popular, and then contrast them with Nanvix. For a more detailed discussion on this topic, we refer the reader to [8], [9], [10], [11].

Minix is one of the first successful instructional OSs introduced by Vrije University [6]. It is fully featured and accompanies a well known textbook, but it presents some drawbacks which turn it inconvenient to be used in OS courses. First, Minix is a large system, when compared to related OSs. Second, besides the practical activities that are presented in Minix's textbook, no teaching methodology is suggested with it. It is up to the instructor to decide how to introduce Minix to students. Finally, it embraces outdated features, such as a segmentation-based paging system with no swapping.

Nachos is an OS, developed at University of California, Berkley [2]. It runs in a simulated MIPS environment as a user-level Unix process, and it was designed so as to provide a minimum working example of a system. Nachos accompanies a series of assignments which are built on top of one another and students work on major portions of the OS. Nachos instructors observed that their system and assignment-based teaching methodology helped students to further consolidate their understanding in OSs. However, they reported that they were not able to find a textbook that covered many of the concepts used in Nachos, thus stepping the learning rate.

GeekOS is an OS developed at University of Maryland, College Park. It is structured as a monolithic kernel and provides the bare-bone features of a modern OS. Based on this system, students work in a series of assignments which are built on the top of one another. An evaluation of GeekOS is presented in [4], and the results showed that students found that GeekOS helped them to understand OS concepts though the course. Moreover, results unveiled that students felt motivated on having GeekOS to run on real hardware.

Table I highlights the main differences between the aforementioned OSs and Nanvix. In contrast, Nanvix is fully featured (complete), and it accompanies a flexible assignment-based teaching methodology that enables instructors to better meet their schedule and teaching needs. Moreover, Nanvix features a Unix System V architecture and Posix compliance, thus making students face a realistic environment. Besides this, Nanvix is smaller, hence making kernel hacking easier; it is shipped with an extensive documentation, thus saving instructor's time; and it runs in virtual machines (easy to debug) and on real hardware (further motivating).

TABLE I
Comparison between Operating Systems.

| Name | Complete? | Teaching Methodology | Comments |
|---|---|---|---|
| Minix | yes | none | outdated features |
| Nachos | no | assignments, not flexible | does not run on hardware |
| GeekOS | no | assignments, not flexible | no Posix compliance |
| Nanvix | yes | assignments, flexible | Unix and Posix support |

## III. The Nanvix Instructional Operating System

Nanvix is an OS that we designed from scratch to be small and simple, and yet fully featured. It originally targets x86-based PCs and features virtual-memory based on paging, a hierarchical Unix file system based on inodes, a uniform device driver interface, and a preemptive priority-based scheduler. Nanvix is under active development, with growing interests on both education and research. This work is grounded on Nanvix 1.2, but all its releases are available at wwww.github.com/nanvix/nanvix. In the paragraphs that follow, we briefly detail the internals of Nanvix. For a more detailed discussion of our OS we refer the reader to [12].

The architecture of Nanvix is outlined in Figure 1. It presents a similar structure to Unix System V [13], and it has been intentionally designed this way because it is adopted in some successful OSs, such as Linux. Nanvix is structured in two layers. The kernel (bottom layer), seats on top of the hardware and runs in privileged mode. Its job is to (i) extend the underlying hardware so that an easier-to-program interface is exported to the higher layer; and (ii) multiplex hardware resources among several users. The userland (top layer), relies on Posix system calls exported by the kernel and it is the place where user software run in unprivileged mode.

The kernel presents a tiny monolithic architecture (7k loc), and it is structured in four subsystems: the hardware abstraction layer; the memory management system; the process manager; and the file system. The hardware abstraction layer interacts directly with the hardware and exports to the other subsystems a set of well-defined low-level routines. The job of the hardware abstraction layer is to isolate, as much as possible, all the hardware intricacies, so that the kernel can easily be ported to other compatible platforms.

The memory manager provides a flat virtual memory abstraction. It does so by having two modules working together: the paging and virtual memory allocator. The former deals with paging, keeping in memory those pages that are more frequently used, and swapping out to disk those that are not. The virtual memory allocator, on the other hand, relies on the paging module to create higher-level abstractions called memory regions, and thus enable advanced features such as shared memory regions, on-demand loading and lazy coping.

The process manager handles creation, termination, scheduling, synchronization and communication of processes. Processes are single-threaded entities and are created on demand, either by the system itself or the user. Scheduling is based on preemption, and in userland it happens whenever a process runs out of quantum or blocks awaiting for a resource. In ker-

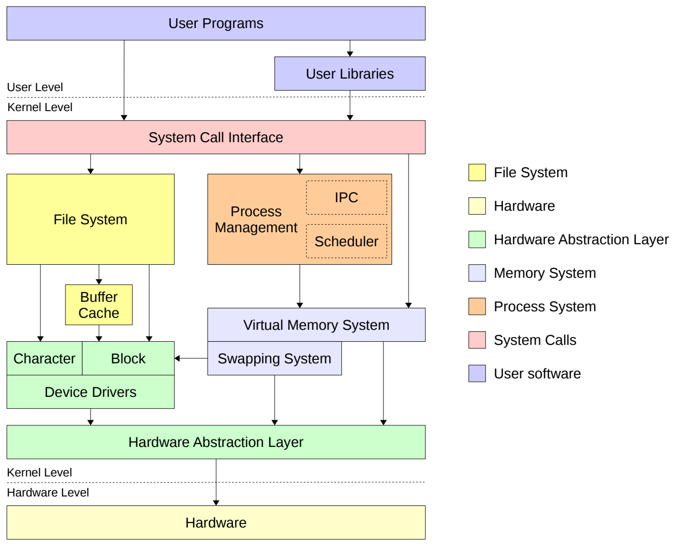

Fig. 1. Nanvix architecture.

nel land, processes run in nonpreemptive mode and scheduling occurs when a processes voluntarily foes to sleep. In addition, the process manager exports inter-process communication facilities, such as Posix pipes and shared memory regions.

The file system provides a uniform interface for dealing with hardware resources. It extends the device driver interface and creates on top of it the file abstraction. Files can be accessed through a unique pathname, and may be shared among several processes. The Nanvix file system is compatible with the one present in Minix, it adopts an hierarchical inode structure, and features mounting points and disk block caching.

## IV. Teaching With Nanvix

In this section, we detail the assigned-based teaching methodology that we propose to be used with our Nanvix. In contrast to related approaches, ours is designed so that instructors may: (i) carry out practical activities in any order they want; and (ii) be free to propose their own activities. First, we present an overview of it, then we introduce the underlying principles that we considered when designing the assignments, and next we present each of them in turn. In the end, we present the evaluation method that we adopted to assess the effectiveness of our Nanvix-based teaching methodoly.

### A. Assignments Organization

Table II presents an overview of the five assignments that are shipped with Nanvix. These assignments are designed for a one semester-course, and may be assigned in arbitrary order. We suggest that the instructor to take about 10-15 minutes every week to discuss with the students what problems they are facing and how they can address them. With respect to grading, we do not pin down a specific scheme, because administrative questions may differ from one course to another. However, we suggest some considerations in the evaluation of assignments.

TABLE II
Assignments shipped with Nanvix.

| Subject | Assignment Name | Activities | Duration |
|---|---|---|---|
| Process Management | Process Scheduling | Implement a priority scheduler | 3 weeks |
| | Semaphores | Implement semaphores in kernel | 3 weeks |
| Memory Management | Page Replacement | Implement page eviction | 4 weeks |
| I/O Management | Prefetching | Implement disk block prefetching | 4 weeks |
| File Systems and Security | Security | Exploit security breaches | 3 weeks |

### B. Assignment Principles

*Improve Existing Features*. Students work improving existing functionalities, rather than building entire system's components from scratch. This way, they can focus more on core design aspects, thus increasing their learning rate.

*Freedom in Design*. Creativity is an important skill for computer scientists. We encourage this by designing assignments in such a way that students are free to choose whatever data structures and algorithms they want.

*Encourage Realistic Solutions*. High performance is an important requirement for an OS and it is usually challenging to deliver. To motivate students to work towards this, each assignment is shipped with a micro-benchmark which they can use to evaluate their solution. We suggest that the more students score on these benchmarks, the more they are rewarded.

*Teamwork*. In industry, software is developed by having tens of small and self-organizing teams to collaborate with one another. To replicate such environment, assignments are designed to be accomplished by a team of 2-4 students.

*Detailed Documentation*. An OS that serves as a supporting teaching tool to theoretical lectures should be well documented, otherwise instructors and students will have difficulties on using it. To address this problem, we provide a detailed documentation of both the kernel and assignments.

### C. Process Scheduling and Semaphore Implementation

In the educational version of Nanvix, processes are scheduled based on a round-robin policy, and no process synchronization mechanism is available. We therefore ask students to (i) implement a priority scheduler; and (ii) implement semaphores in kernel land and export this feature through the system call interface to userland. Students are free to choose their own design. However, they should follow the system specification for the semaphore implementation, which is compatible with Posix, and should benchmark their solutions by running a Producer-Consumer example that is provided.

In this assignment, students learn how the OS schedules processes to execution, how complex scheduling policies may be built on top of simpler ones, and how synchronization primitives may be implemented. In addition, students learn the importance of dynamic priority scheduling; have insights about how to design fair semaphores; and learn how to add new system calls to an OS's kernel.

Students should have about six weeks to work on this assignment, three weeks for each part. With respect to grading, we suggest the instructor to reward with extra points those students that have implemented a dynamic priority scheduling policy in the first activity, and semaphores with access verification and/or fair scheduling in the second one.

## D. Page Replacement

In the educational version of Nanvix, pages are evicted from main memory and swapped out to disk in a first-in-first-out fashion with a local-replacement policy. Such strategy is simple to implement, but it leads to a poor performance when compared to other strategies. To solve this problem, students are asked to implement a more efficient page eviction algorithm. To encourage them to work on realistic solutions, we require students to evaluate the performance of their strategy with a matrix multiplication benchmark that we provide.

With this assignment, students learn important principles on system design. First, they further understand how the OS provides the illusion of a large memory to users, through the virtual memory and paging techniques. Second, students work deepen their knowledge on how the paging system works internally. Finally, students learn how a smart eviction strategy may lead to a good system's performance.

Students should have about four weeks to work on this assignment: one week to first understand how the existing paging system works, two weeks to implement a more efficient page eviction algorithm, and another week to perform benchmark evaluation and final adjustments on their solution. With respect to grading, we suggest the instructor to reward students according to the robustness of their solution.

## E. I/O Prefetching

In Nanvix, the disk buffer cache features synchronous reads and asynchronous write operations, but it lacks prefetching support for sequential reads. To further enhance the file system's performance, we ask students to implement this latter feature. To accomplish so, they have to add a read ahead function in the buffer cache, then they have to hack the `read()` system call to make use of this new operation; and finally they have to benchmark their solution.

With this assignment, students learn several lessons. First, they learn how prefetching can boost the performance of the system. Second, while working with the the buffer cache, students will have to deal with race conditions, and thus exercise their previous knowledge on process synchronization. Finally, students work with a core subsystem of the device system, thereby further understanding how it works as a whole.

Students should have about four weeks to work on this assignment: one week to first understand how the buffer cache subsystem works, two weeks to implement the block read ahead operation, and another week to hack the `read()` system call and evaluate their solution. With respect to grading, we suggest instructors to reward students according to the simplicity of their solution – the solution should take no more than a hundred lines of code.

## F. Exploiting Security

In this assignment we students to seek and exploit security breaches in Nanvix. Intentionally, five vulnerabilities were left open in the system: (i) password cracking by brute-force attacking the passwords file; (ii) privilege escalation by editing the unprotected system's initialization file; (iii) kernel panic by writing random content in memory with the `read()` system call; (iv) memory dump using the `write()` system call; and (v) denial of service with a fork-bomb attack.

Students should browse the system call interface of the file system, and they learn two topics on information security with a hands-on cracking approach. First, they understand how vulnerabilities may be exploited and compromise sensitive information. Second, they are encouraged to rethink about software design, in what concerns information security.

What concerns course schedule, students should have about three weeks to work on this assignment: one week for learning how Nanvix deals with security, another week for seeking and exploiting security breaches, and a final week for writing reports about the encountered vulnerabilities. Regarding grading, we suggest instructors to reward students according to the number of security breaches they have exploited.

## G. Evaluation Method

In order to evaluate the teaching methodology that we propose, we applied it in the undergraduate OSs course of PUC Minas. We measured the performance and motivation of students by observing their score on exams and assignments, and compiling their feedback from emails and surveys that we distributed, respectively.

At PUC Minas, undergraduate computer science students take the OSs course in their fifth academic semester. In this course that lasts about 16 weeks (64 hours), lectures are strictly theoretical and practical topics such as shell and concurrent programming are covered in extra-class assignments. To introduce our methodology, we adopted the following method.

In a first academic year, we kept the aforementioned teaching approach, but we gathered statistics about the performance of students on exams to find out what were the main challenges that they faced. Then, in a second academic year, we used this information to apply a slightly different teaching approach, in which we kept traditional lectures, but we adopted assignments in Nanvix. Again, we gathered statistics about what difficulties students experienced on exams. However, in addition, we asked students to answer a short survey about their thoughts on Nanvix being used for practical assignments..

Finally, in a third academic year, we used our previous experience to re-redesign the OSs course at PUC Minas. In this new version of the course, we kept the normal course schedule and assignments in Nanvix, but additionally we reserved some lectures to introduce the internals of our OS, and we allocated a partial-time teaching assistant to help students with their questions. In order to evaluate the effectiveness of this new version of the course, we gathered statistics about the performance of students on exams, and we also asked them to answer the same survey from the previous academic year.

## V. Teaching Methodology Evaluation

In this section, we discuss the results that we observed when we applied our teaching methodology in the Operating Systems course of PUC Minas. First, we discuss the background experience we had when introducing our Nanvix-based methodology, and then we move on to an evaluation of the assignments and the teaching methodology itself.

### A. Background Experience

In the first academic year, we observed that students had difficulty on the process and memory management topics. Their average score on exams in these subjects were 53.9% and 51.1%, respectively. To address this problem, in the second academic year we applied students to the Process Scheduling, Page Replacement and I/O Prefetching assignments in Nanvix. With this redesign, we observed a 68.1% and 59.1% raise in the average student score, in the process and memory management subjects respectively.

To redesign the OSs course for the third academic year, we considered the performance of students on exams in the previous academic year and their answers to a survey that we distributed. The answers to this survey are outlined in Figure 2. According to the survey, about 80% of the students answered that our assignment-based approach helped them to better understand some concepts and principles that were introduced in the lectures. Additionally, about 35% of the students reported that Nanvix helped them in practical topics. When we asked students to give us their thoughts about our assignment methodology, about 50% of them asked us for practical lectures on Nanvix, 20% suggested us to introduce our OS in theoretical lectures, and about 45% asked for the Security assignment.

Based on this previous experience, in the third academic year, we applied to students the Process Scheduling, Semaphores, and Security assignments. Moreover, we alternatively also let students work on the Page Replacement assignment for extra points. In this redesign, we found out that (i) Nanvix helped 83.8% and 40.5% of the students, in theoretical and practical topics, respectively; and (ii) that 5.6% and 13.9% of the students complained about the schedule and the difficulty of assignments, respectively.

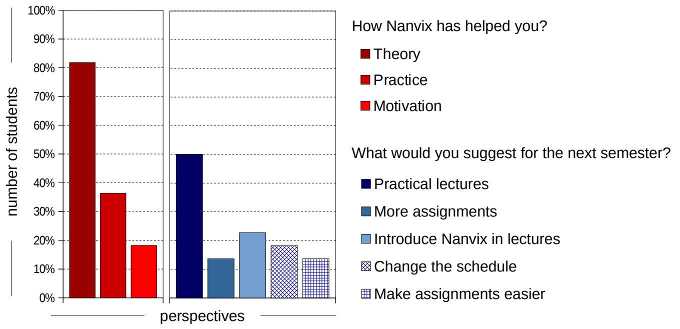

Fig. 2. Survey results for second academic year.

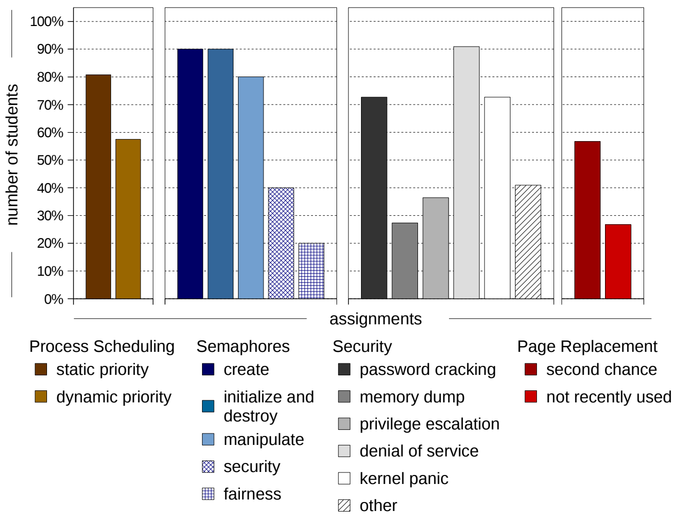

Fig. 3. Assignments evaluation detailed.

### B. Assignments Evaluation

Figures 4 and 3, presents the score of the students in the assignments. These results were gathered during the third academic year from a sample of 44 students.

In the Process Scheduling assignment 34 students submitted their solutions on schedule. In average, 81% of them completed the requested tasks, and 58% implemented a priority-based scheduler. They reported that the difficulty of the assignment was adequate, and the hardest part was to understand how priorities were assigned by the kernel.

In the Semaphores assignment, 15 students submitted their solutions on schedule. In average, 90% them were able to successfully complete the requested tasks, and 30% of have finished the extra ones. Students reported that the assignment was challenging, but it helped them to understand how the kernel manipulates the state of processes under the hood. Concerning the difficulty of the assignment, students found that the hardest part was to write the system call for manipulating the semaphore itself. According to them, at first it was not clear how to properly put processes to sleep and wake them up, but once they figured out that the process management module exported routines for doing so, these tasks became straightforward.

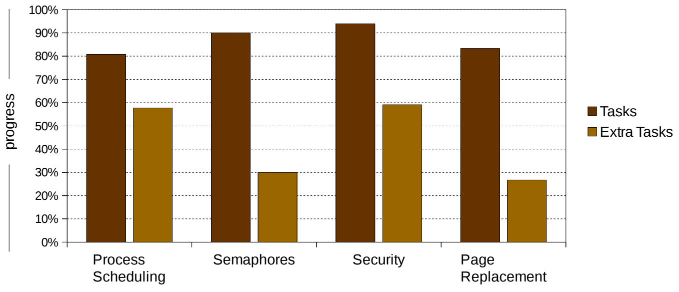

Fig. 4. Assignments evaluation overview.

TABLE III
COURSE EVALUATION.

| Academic Year | Failure Rate | Average Score |
|---|---|---|
| 1st year | 53.3% | 57.9% |
| 2nd year | 16.0% | 67.6% |
| 3rd year | 27.9% | 65.2% |

In the Security assignment, 31 students submitted their solutions on schedule. In average, 93.9% of them were able to find at least three vulnerabilities in the system, and 59.1% succeeded on exploiting more than three vulnerabilities. Students reported that the assignment was challenging, but fun. They enjoyed to deal with information security with a hands-on approach, and according to them, the *privilege escalation* and *memory dump* security breaches were the hardest to exploit.

In the Page Replacement assignment, 10 students submitted their answers on schedule. In average, 83.3% were able to successfully complete the requested task. The most popular solution among the students was the *second chance* algorithm, and 26.7% of them implemented the *not recently used* page replacement policy. They found that the difficulty of the assignment was adequate, and that the hardest part was to know how to correctly inspect page table entries.

### C. Course Evaluation

During the three academic years we measured the performance and motivation of students. These results are presented in Table III and discussed next. In a long-term perspective, our teaching methodology helped students to enhance their performance in the OSs course. With Nanvix, the average score of students increased in 11.2% and the failure rate dropped in 47.7%. Furthermore, the number of students who scored more than 75% increased from 7% to 21% along these academic years, which shows that our Nanvix-based methodology motivated students to work hard and also to push up their score in the course.

Additionally, during the two last academic years, several students reported us that they enjoyed working with Nanvix. Figure 5 presents a compilation of the feedback that we received from students on emails, surveys and exams. In this chart, keywords that have higher counts appear bigger than those that have lower counts. Students found that Nanvix has helped them to consolidate principles and concepts that they studied in theoretical lectures, learn about OS's internals, and fell further motivation and interest for the field.

## VI. CONCLUSIONS AND FUTURE PERSPECTIVES

In this work, we (i) present Nanvix, an OS to be used in OSs courses as supporting software; (ii) introduce a flexible assignment-based teaching methodology that may be used with it; and (iii) we presented an evaluation of this methodology when applied in the OSs course in PUC Minas.

To introduce our teaching methodology, we adopted an incremental three-year method. In the first one, we kept traditional lectures and assignments; in the second year, we kept traditional lectures, but we adopted assignments in Nanvix; and finally in the third academic year, we introduced Nanvix in theoretical lectures and applied assignments on it. With our teaching methodology, the average score of students in the OSs course increased in 11.2%, and the failure rate dropped to 47.7%. Moreover, students found that Nanvix helped them to consolidate OS principles and felt more motivated.

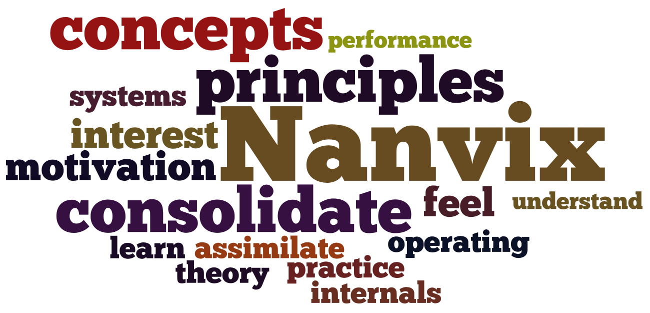

Fig. 5. Qualitative evaluation of Nanvix.

Besides PUC Minas, Nanvix is already in use in the Federal University of Santa Catarina (Brazil) and University of Grenoble Alpes (France). In future work, we intend to extend and improve our teaching methodology based on the experience of instructors of these partner universities.

## REFERENCES

[1] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[2] W. A. Christopher, S. J. Procter, and T. E. Anderson, "The nachos instructional operating system," in *USENIX Winter 1993 Conference Proceedings*, ser. USENIX'93. Berkeley, CA, USA: USENIX Association, 1993.

[3] D. A. Holland, A. T. Lim, and M. I. Seltzer, "A new instructional operating system," in *Technical Symposium on Computer Science Education*, Cincinnati, Kentucky, USA, 2002, pp. 111–115.

[4] D. Hovemeyer, J. K. Hollingsworth, and B. Bhattacharjee, "Running on the bare metal with geekos," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '04. New York, NY, USA: ACM, 2004, pp. 315–319.

[5] B. Pfaff, A. Romano, and G. Back, "The pintos instructional operating system kernel," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09. New York, NY, USA: ACM, 2009, pp. 453–457.

[6] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems: Design and Implementation*, 3rd ed., ser. The MINIX Book. Pearson Education International, 2009.

[7] C. M. da Costa, J. L. Fragoso, L. Murliky, L. da Luz Silva, A. Fracalossi, C. Brasil, G. Debom, and R. Matias Jr, "Nke-um nanokernel educacional para microprocessadores arm," 2014.

[8] C. L. Anderson and M. Nguyen, "A survey of contemporary instructional operating systems for use in undergraduate courses," *J. Comput. Sci. Coll.*, vol. 21, no. 1, pp. 183–190, Oct. 2005.

[9] S.-w. Hwang, "Blended learning for teaching operating systems with windows," *SIGCSE Bull.*, vol. 41, no. 3, pp. 380–380, Jul. 2009.

[10] O. Laadan, J. Nieh, and N. Viennot, "Structured linux kernel projects for teaching operating systems concepts," in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 287–292.

[11] R. Cox, M. F. Kaashoek, and R. Morris, "Xv6, a simple unix-like teaching operating system," 2011.

[12] P. H. Penna, "The Nanvix Operating System," Pontifical Catholic University of Minas Gerais, Tech. Rep., mar 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01495741/

[13] M. J. Bach, *The Design of the UNIX Operating System*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.