A Scalable Operating System Experiment Platform Supporting Learning Behavior Analysis

Lei Wang, Ziqi Zhen, Tianyu Wo, Bo Jiang, *Member, IEEE*, Hailong Sun, *Member, IEEE*, and Xiang Long

***Abstract*—*Contribution:* The design of an operating system (OS) experiment course with a gentle learning curve is proposed and a scalable OS experiment platform supporting learning behavior analysis is presented.**

***Background:* In the teaching practice of the OS experiment course, several problems were faced. First, the learning curve for the students is too steep. Second, the OS experiment is hard to scale to a large number of students. Finally, it is difficult to track the learning behaviors of the students and to provide feedback to the teaching plan.**

***Intended Outcomes:* By smoothing the learning curve and providing targeted and rapid feedback, students using the system can more effectively master the knowledge of the OS by completing experimental tasks.**

***Application Design:* A large number of students can log in to the experiment system and complete multiple labs online. In addition to using automated testing to provide fast feedback, the lab system also collects learning behavior data of the students, enabling the teachers to analyze the data and adjust their teaching plan accordingly.**

***Findings:* The performance and learning-behavior data of the students from 2015 to 2018 showed that the OS experiment is effective to scale to a large number of students with satisfiable teaching effectiveness.**

***Index Terms*—Learning behavior, operating system (OS) experiment, scalability.**

I. Introduction

OPERATING system (OS) is one of the core courses for computer science and engineering-related majors. It is closely related to four knowledge areas proposed in the Computer Science Curricula 2013 by ACM/IEEE-CS Joint Task Force on Computing Curricula: system fundamentals (SFs), parallel and distributed computing (PD), information assurance and security (IAS), and platform-based development (PBD) [1]. While the OS theory courses familiarize students with the main concepts, the OS experiment can help students to understand the OS concepts at a deeper level by building an OS by themselves [2]. Therefore, many top computer science departments have attached great importance to the experiment design of OS courses [3]–[10].

The OS experiment system proposed in this article was initially ported from the JOS experiment system [9] used in MIT to MIPS architecture. The JOS OS was used by MIT in the lab of the course *OS Engineering* [2]. The porting, begun in 2007, took several years to complete. In 2013, 20 top students were selected to join the course and they all got excellent results. In this class, most of the students completed all the labs with good performance. However, when all the sophomore students in the department (with about 255 students) joined the experiment course in 2014, only less than 25% of the students successfully completed all the labs. An analysis of the teaching process revealed the following challenges faced by the OS experiment course.

First, the learning curve for most students is too steep. When the number of students is large, the students naturally differ in terms of ability and attitude. For those top students, completing the OS experiment is never a big problem. However, for most students, it is important to ensure the learning curve is not too steep so that they can keep up to complete the labs.

Second, it is difficult to scale the OS experiment to a large number of students. When the number of students grows large, providing a scalable online OS experiment platform and evaluating the code of the students in each lab becomes nontrivial. A scalable OS experiment platform must provide an integrated development environment for the growing number of students. Furthermore, within an online experiment platform, evaluating the code submitted by a large number of students must be performed in a timely manner.

Third, it is difficult to get feedback from the learning process of the students and adjust the teaching plan accordingly in a short time. This is due to the lack of mechanisms in the experimental environment to gather data on the students' learning behavior.

In previous work, many effective OS experimental platforms were proposed to help students learn OS by *doing* [4]–[10]. While most of the OS experiment courses corresponding to these platforms provide detailed lab guidance or special assistant software [5] to help students cope with the steep learning curve, they were not specially designed to scale the experiment course to a large number of students. Furthermore, although learning behavior analysis for teaching improvement is extensively studied in previous work [11]–[15], these works were not targeted at teaching improvement of the OS experiment course.

Manuscript received August 15, 2019; revised December 13, 2019 and February 6, 2020; accepted February 15, 2020. This work was supported in part by the NSFC of China under Project 61672073, Project 61772056, and Project 61690202, and in part by the Teaching Reform Funding of Beihang University. *(Corresponding author: Bo Jiang.)*

The authors are with the School of Computer Science and Engineering, Beihang University, Beijing 100083, China (e-mail: wanglei@buaa.edu.cn; yradex@buaa.edu.cn; woty@act.buaa.edu.cn; jiangbo@buaa.edu.cn; sunhl@buaa.edu.cn; long@buaa.edu.cn).

Digital Object Identifier 10.1109/TE.2020.2975556

In this article, to facilitate learning among average students, the six labs within the OS experiment are carefully designed so that they are easier at first but gradually become more and more challenging. The guidebook for the experiment is also carefully designed to guide students step by step to complete the experiment. To scale the OS experiment to a large number of students, a VM-based integrated development environment within a private cloud is proposed for the students to program, test, and debug. Furthermore, to reduce the code submission and code evaluation overhead, a Git server and an automatic code evaluation system are integrated into the experimentation platform so that the dispatch of labs and the evaluation of the code submitted by the students can be performed automatically within a short time. Finally, to get feedback from the learning process of the students, the behavior data of the students are also collected within the integrated OS experiment platform to facilitate analysis and continuous improvement.

The contributions of this article are twofold. First, an integrated and scalable online OS experiment platform that enables learning behavior analysis is presented. Second, based on the learning data of 878 students from the OS experiment course over four years, the teaching effectiveness of the OS experiment course is thoroughly evaluated.

The organization of the remaining sections is as follows. Section II presents the design of the OS experiment course, which tries to make the learning curve of the course gentle. Section III discusses how to scale the experiment to a large number of students. Section IV presents the learning behavior tracking system. Section V evaluates the overall teaching effectiveness with a large-scale case study over four years. Finally, Section VI presents related work and is followed by the conclusion in Section VII.

II. Making the Learning Curve Gentle

A. Curriculum Design

To make the learning curve of the OS experiment gentle, the six labs are carefully designed with increasing but controllable difficulty. As the students go through the labs, they will gradually build a small but complete OS.

In general, the students are asked to develop a small OS on the MIPS platform. It includes six labs as shown in Fig. 1. The details of lab 1–lab 6 are as follows.

1) *Lab 1 (Boot and System Initialization):* The student will analyze the hardware boot process to understand the linking, loading, and relocation of the OS kernel. They will then implement the *printf* API via a serial port to understand the core OS submodules.
2) *Lab 2 (Memory Management):* The student will understand the layout of the MIPS memory and implement the physical and virtual memory management module.
3) *Lab 3 (Process Management):* The student will implement clock interrupt handler and process management functions.
4) *Lab 4 (System Call):* The student will understand the system call mechanism on MIPS and implement several system calls.

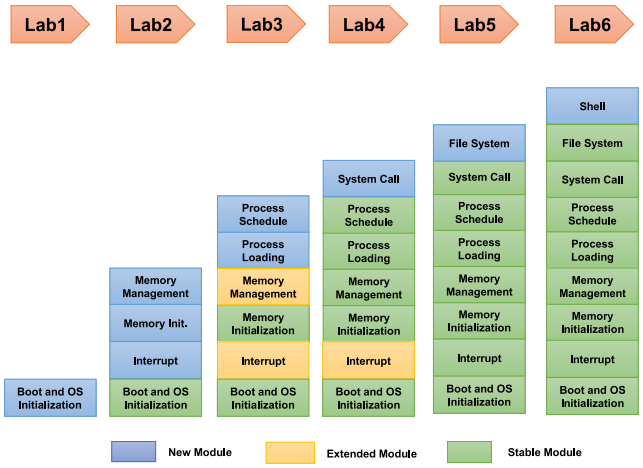

Fig. 1. Design of OS experiment.

5) *Lab 5 (File System):* The student will implement a simple file system.
6) *Lab 6 (Shell):* The student will implement a basic shell and combine the work of all six labs to form a small OS.

Note that lab 5 and lab 6 are not mandatory to pass the whole course. However, if students want to get a high score for the whole course or if they have not got scores in previous labs sufficient to pass the course, they must try their best to perform well in lab 5 and lab 6. As shown in Fig. 1, the modules in blue are the new modules to implement, the modules in yellow are the old modules to extend with more functionality (e.g., interrupt in lab 4), and the modules in green are modules that remain unchanged in the corresponding lab. Each lab contains two parts. One part is the before-class test; the other part is the in-class test. All students must first pass the before-class test (i.e., with scores higher than 60) before they can take part in the in-class test. The before-class tests are more like homework which students are encouraged to complete independently; the in-class tests are more like exams, which students must finish independently in class. The setup of the two tests is to incentivize each student to finish the experiment independently. If a student consults other students to finish the before-class test, he will likely fail the in-class test.

The design of the experiment follows two basic principles. First, the experiment follows an iterative development model. The first few labs will build the core OS modules with key functionality. More advanced features are added in the follow-up labs to gradually form a small but complete OS. In this way, the student will not be overwhelmed at the beginning by the complexity of the OS. Second, the experiment course sets up different goals for students with different capability. For the in-class tests, several basic tests with moderate difficulty are set up within each lab. The majority of the students need to finish only the basic tests. In addition to the basic tests, ambitious students get bonus points by finishing some challenging extra tests.

B. Step-by-Step Guidebook

To further help the students to build the small OS and make their learning curve gentle, a step-by-step guidebook [16] for

the OS experiment is carefully composed. Following this book, the students get the most important and relevant guidance they need in time within each lab.

The idea of providing a step-by-step guidebook for the beginners in a new computer science course is also emphasized by the work of Guzdial [17]. In particular, they maintain that "putting introductory students in the position of discovering information for themselves is a bad idea." Furthermore, Otero and Aravind [18] also suggested providing a step-by-step guidebook to help the student build the minimal embedded OS (MiniOS) from the ground up. They argued that the challenges of the OS experiment course are substantially different from what students have encountered in preceding courses and "most students lack the experience, the patience, and the right approach to meticulously construct and debug low-level systems' code."

Within the course of this article, the students have similar experiences. The students lack experience with the Git tool, the UNIX programming environment, the programming of low-level system code on MIPS architecture, and debugging of the system code. Without proper guidance, many students easily get lost during the labs. A comprehensive step-by-step guidebook is therefore valuable to help them to overcome these challenges. Since the design of the labs changes each year (to prevent plagiarism), the guidance book must be updated accordingly. To keep the content of the guidebook up to date, the electronic version is dynamically published through a central Web portal of the OS course.

III. Scaling the OS Experiment Platform

To scale the OS experiment to a large number of students, an integrated online experiment environment is proposed. The integrated online experiment environment combines the virtual machines, version control system, and the automatic code evaluation system within a private cloud to achieve scalability. From the perspective of the students, the initial code release, coding, compilation and execution, code submission, code evaluation, and debugging are performed within the integrated online experiment environment.

A. Overview of the OS Experiment Platform

In this section, an overview of the OS experiment platform is presented. As shown in Fig. 2, the platform consists of the following parts.

1) The virtual machine platform includes Linux OS, the cross-compiler toolchain, and the MIPS simulator.
2) The Git server maintains a separate account for each student. The initial code and the submitted code are all maintained by the Git system.
3) The jump servers act as the portal for students to log in to the experiment platform. The student must first log in to the jump server before they log in to the VMs. On the one hand, the jump server provides one layer of security such that the VM will not be exposed directly. On the other hand, the jump server is an ideal place to track the learning behavior of the students.

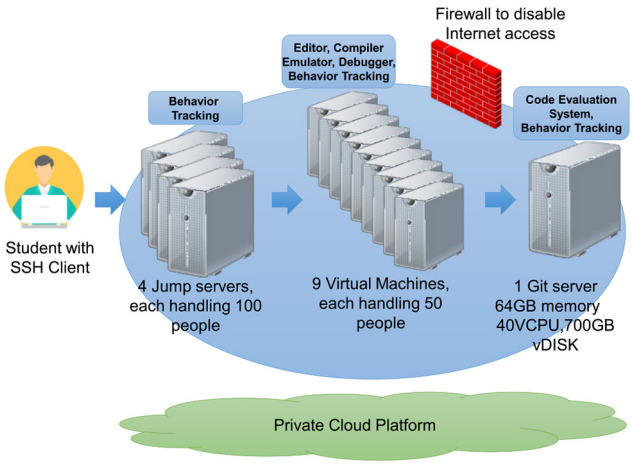

Fig. 2. Integrated environment structure of OS experiments.

4) The automatic code evaluation system performs testing on the submitted code to check for correctness. For each task in each lab, a set of test cases is designed to check against the submitted code. The evaluation of the submitted code is automatically triggered by code submission with a Git hook script deployed on the Git server.
5) The learning behavior tracking system collects the students' behavioral data during the experiment, and it is distributed across the jump server (since 2018), the VMs (2016–2017), and the Git server.

B. Virtual Machine Platform

The virtual machine platform provides students with the environment and toolchains for coding, compilation, execution, and debugging. Within the virtual machine, a Linux system, the Vi/Vim editors, debuggers, the Gxemul simulator, and the GCC cross-compiler for MIPS are setup for use. Each student has an independent account and a working directory within the Linux system on the virtual machine. The students log in to the Linux system with an SSH client to do the experiments. The working directory of the students is securely separated to prevent plagiarism.

To solve the scalability problem, the virtual machine provides the students with a virtualized programming environment while the Gxemul emulator provides the students with an emulated execution environment. Therefore, the virtualization and emulation techniques essentially decouple the experiment platform from the specific hardware system. In this way, the OS experiment platform can be scalable with the resources available.

C. Git Server

The Git version control system is crucial to connect coding and evaluation. Each student has their own code repository, which is not accessible by other students. Each test (i.e., coding task) within a lab has its own branch. To start, each student checks out the branch corresponding to the test to work with. Then the student can code, compile, and test in their local directory. When the student finishes the coding task, they can

submit it to the corresponding branch in the Git repository. Upon submission, a Git hook script is triggered for execution to perform automatic code evaluation. If the submitted code passes the tests, the Git server will automatically push the next task to the student.

D. Automatic Code Evaluation Tool

Providing feedback to the students on assessment tasks is useful to improve their performance [19]. Evaluating the submitted code from so many students is also a time-consuming activity. To make the OS experiment platform scalable to a large number of students, it is crucial to make the code evaluation task automatic.

When students submit their code through the Git server, the code evaluation system is automatically triggered by a Git hook script to obtain the code submitted from the students. Then, the code evaluation system starts compiling and testing the submitted code automatically with predefined test cases. When the testing ends, the system generates a test report and returns the evaluation results back to the student. If the code passes the tests, the system automatically pushes the next task to the student. Note: the score of a task is the sum of scores for all the passing test cases. The total score for a task is 100; a student passes the task if they have a score higher than 60.

IV. Learning Behavior Tracking System

To collect and analyze students' behavioral data, a learning behavior tracking system is built into the integrated environment. Based on the learning behavior data collected, learning behavior analysis can be performed. Finally, the teaching plan can be adapted based on the analysis results.

As discussed in the previous section, the learning behavior tracking system is distributed across the jump server (since 2018), the VMs (2016–2017), and the Git server. The collected data mainly contain three parts.

1) *Login Activity and Command History:* The collected data include the timestamps of the activities of the students, account information, and all user input and output. Before 2018, the login information and command history were collected within the virtual machines where students logged in. Since the students used the SSH clients to log in to the VMs, the SSH daemon was modified to collect the login information. Since 2018, students must first log in to the jump server before logging in to the virtual machines. The login activity and command history have been collected within the jump server since 2018.
2) *Git Log:* All Git operations used to interact with the Git server by the students are recorded within Git log.
3) *Code Evaluation Report:* After a successful code submission through Git, a hook script is triggered to evaluate the code submitted. The code submission information and the evaluation results are recorded within the code evaluation report.

V. Evaluation

To evaluate the effectiveness of the proposed OS experiment system as well as the course design with educational research methodologies, a large-scale case study was conducted based on the OS experiment course across four years (2015–2018). The data on the learning behaviors and students' scores were collected and analyzed to evaluate the proposed OS experiment system and course.

TABLE I
Number of Students at Different Levels

	90-100	60-89	0-59	Total
2015	19 13.1%	124 85.5%	2 1.4%	145
2016	25 17.4%	116 80.6%	3 2.1%	144
2017	54 19.2%	191 67.7%	37 13.1%	282
2018	70 22.8%	193 62.9%	44 14.3%	307

A. Setup of the Case Study

The case study contains two main parts. For the first part, the scores of all students were tracked from 2015 to 2018. Then, the performance of the students across four years was analyzed to evaluate the overall teaching effectiveness of the course. For the second part, the case study analyzed the detailed learning behavior data across four years. Then, the case study tried to find trends within the data as well as correlations between learning behavior and score. To measure the learning performance of the students, the students are classified into three categories based on their score: 1) excellent (90–100); 2) average (60–89); and 3) underperforming (0–59). This classification is also widely used in many universities in China for grading.

The limitation of this case study is that it was conducted solely within one university. A more comprehensive study conducted across several universities using the proposed platform may further strengthen the validity of the findings.

B. Evaluation of Overall Teaching Effectiveness and Scalability of the Experiment Platform

To evaluate the overall teaching effectiveness, the number of excellent, average, and underperforming students in four years was collected and presented as shown in Table I. The ratio of excellent students increases year by year. Although the number of students doubled in 2018, the students graded as excellent still reached the highest ratio. The number of underperforming students also increased in 2017 and 2018.

There are two reasons for this. First, the difficulty of the experiment has gradually increased since 2017. Second, more students from other engineering majors (i.e., not computer science) had the option to select the OS experiment course since 2016. In 2016, the number of students from other majors was only 24 while in 2017, the number increased to 75. Due to their relatively weak background in computer science, the percentage of underperforming students among them is relatively higher. In general, despite the increase in the difficulty of the

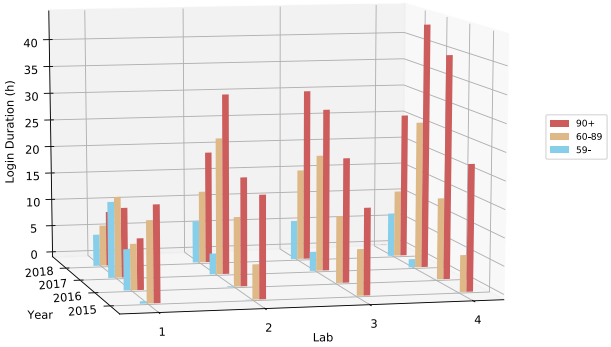

Fig. 3. Average login duration of students with different scores.

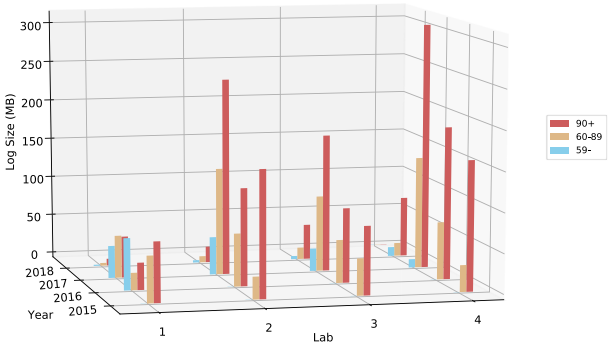

Fig. 4. Average log size of students with different scores.

labs and the number of students over the years, the number of excellent students gradually increased.

C. Learning Behavior Analysis

1) Analysis of Login Duration: Fig. 3 demonstrates how long the students with different scores logged in to the integrated experiment environment in various labs from 2015 to 2018. The *x*-axis shows the first four labs while the *y*-axis shows the login duration in hours. Only the data from the first four labs is presented because the number of students completing lab 5 and lab 6 was small in 2015 and 2016. In general, within the labs of each year, the login durations of the top students are consistently higher than those of the average students, which in turn are consistently higher than those of the underperforming students.

Students' working time (working on labs) and idle time (doing nothing while online) are further analyzed. Compared with the average and underperforming students, those top students not only spent more time in working but they also spent more time thinking or daydreaming (i.e., idle)! Accordingly, the students were encouraged to spend more time on the experiment platform to improve their performance in the course.

2) Analysis of Log Size: The log sizes for students with different grades in different labs are shown in Fig. 4. Because the number of students completing lab 5 and lab 6 was small in 2015 and 2016, only the data from the first 4 labs are presented.

In general, from 2015 to 2017, the online log of the students keeps growing, indicating the students completed more work in the experiments. The log size in 2018 reduces significantly as a different logging scheme is adopted. Instead of using SSH daemon, the jump server is used to perform logging since 2018, which is more concise.

Within the labs of each year, it is noteworthy that the log sizes of excellent students are often more than two times that of average students. In contrast, the log sizes for the underperforming students, in general, are much less than that of the average and top students. Therefore, the size of log generated by a student seems to reflect the effort they spent on the experiment, which may directly impact their performance. Within the course, the log sizes of the students were monitored. Those students with small log size were encouraged to work harder.

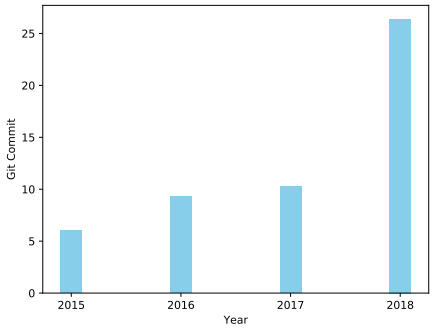

Fig. 5. Average Git commit count.

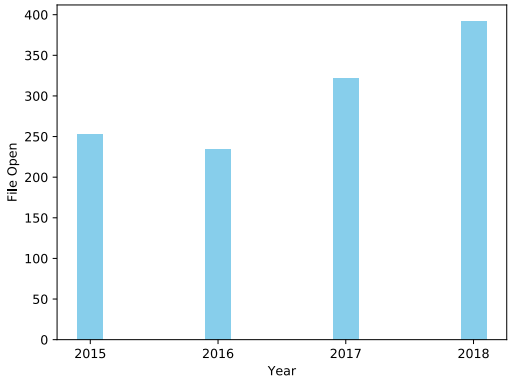

Fig. 6. Average file open count.

3) Git Commit Count Over Years: As shown in Fig. 5, the *x*-axis shows the years while the *y*-axis shows the average Git commit count of the students. It seems that the average Git commit count increases steadily over the years. A further investigation of the submitted code shows that the students are practicing the "code a little/test a little" programming philosophy within the experiment platform. Most of the students write some new code, then perform a Git commit to trigger the automatic code evaluation system to help check errors within their code. Therefore, the automatic code evaluation system is also helpful to cultivate their programming habits.

4) File Open Count Over Years: The average file open count of the students over the years is shown in Fig. 6. Despite a small decrease in 2016, the average file open count for a student gradually increases over the years in general. The increase in the average file open count shows that the

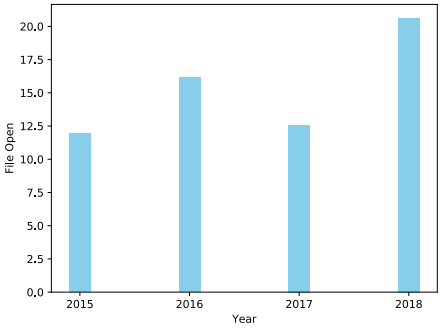

Fig. 7. Average architecture-related file open count.

students tend to read or modify more OS code over the years, which also reflects the increase in the difficulty of the labs.

5) File Open Count of Architecture-Related Files: The file open count for architecture-related files (bars on the right) is shown in Fig. 7. These are mainly files written in assembly languages related to the underlying MIPS architecture. In 2015, the students read only a few architecture-related files. Since these architecture-related files are believed to be important for the student to understand the OS implementation, several questions on architecture-related OS implementation are designed [20]. The students are asked to answer them in their experiment report and in the oral defense when applying for an excellent grade in 2016. As a result, the students read more architecture-related files to answer these questions in 2016. In 2017, similar questions are also raised as homework. However, the students in 2017 got answers to some of these questions from students in 2016 (which is hard to prevent in homework). Therefore, the teaching plans are adjusted in 2018 by designing tests that involved modifying architecture-related files [21]. As a result, the students read more architecture-related files in 2018.

VI. Discussion

Tanenbaum and Woodhull [3] first proposed to learn the principles of OS together with the implementation techniques with the MINIX OS. Stanford University proposed the Pintos [4] OS for OS projects. Their projects also provided an integrated online environment for programming and automatic testing. Different from the OS labs of this article, their projects were done by a team of students and their OS experiment system did not highlight learning behavior tracking abilities. The CALEE [5] system provided a self-learning assistant (SLAT) and a collaborative learning website (CLW) to help the students learn OS with limited teaching resource. The xv6 [6] OS was initially developed by MIT as a code companion to help teach OS principles while the JOS [9] was used for labs. In 2019, MIT changed the OS for its lab from JOS to xv6. In 2015, Otero *et al.* proposed to design OS projects with the objective of developing a MiniOS [7] from scratch. Ziwisky *et al.* [8] proposed an educational OS for multicore architecture. In [10], the University of Washington asked students to build a small but the functional OS (Nachos OS) from scratch. Subsequently, students ran a distributed application on the network-enabled OS.

Learning behavior analysis is also valuable for teachers to improve their teaching effectiveness. Many previous works have studied the programming/learning behavior of students based on different sources of data, such as programming assignments [11], IDE/Web browser logs [12], and newsgroup discussions [13]. In particular, Cleaver and Elbasyouni [14] proposed to monitor the behavior of students when learning through interactive online tutorials. They used Web-based parameter-passing strategies and cookies to help record the learning behavior of students. Li and Tsai [15] analyzed the learning behavior of students when using online learning materials within a mobile phone programming course. They monitored the students' time spent accessing online learning materials to classify students into different clusters. They found that those students who infrequently used any learning materials demonstrated lower motivation and learning performance. In a comparison of students who intensively used all learning materials and students who intensively used only lecture slides, they found the former students had higher homework scores but both groups had similar final examination scores.

An online platform is also useful to improve the teaching effectiveness of labs. Arias *et al.* [22] proposed the COSTE generic tele-education system for OS teaching. COSTE provided online courses, student training with practical components, queries to the teacher, semiautomatic assessment, and an automatic validation mechanism. Hassan *et al.* [23] proposed an online laboratory for computer science education using virtualization technologies. Jong *et al.* [24] proposed the use of game-based cooperative learning in an OSs course. They developed an online game to enable students to learn cooperatively. They found that the desire to win the game motivated the students to learn from online course materials before playing the game, which in turn helped them achieve better learning outcomes.

With the help of the Internet, massive open online courses (MOOCs) are also gaining popularity these years [25], [26]. An online experiment platform based on the Internet is also promising since high-quality experiment teaching resources can be shared across geographical boundaries.

While most of the existing OS experiment platforms discussed above provide effective guidance to help students write OS code, they are not explicitly designed to scale the experiment course to a large number of students. In particular, the OS experiment platform proposed in this article provides a scalable integrated development platform for programming and code evaluation. Furthermore, while there are already many systems to track the learning behavior of students during lectures or labs, the platform proposed in this article aims at providing effective learning behavior tracking within an online OS experiment platform.

Therefore, the main contribution of this article is to propose the integrated and scalable online OS experiment platform supporting learning behavior analysis. Furthermore, the teaching effectiveness of the OS experiment course is also thoroughly

evaluated based on the learning data of 878 students over four years.

VII. Conclusion

Effective teaching of the OS experiment course faces several challenges. First, the learning curve is steep for the students. Second, it is difficult to scale the experiment to a large number of students. Third, it is difficult to gather the learning behaviors of the students, which makes it impossible to provide feedback to the teaching plan.

To address the above problems, the labs within the OS experiment course are carefully designed so that the learning curve for students is gentle. Furthermore, a scalable OS experiment platform is proposed, which can easily support a large number of students with virtual machines and automatic code evaluation techniques. Finally, the OS experiment platform can also monitor the learning behaviors of the students, which can effectively provide feedback to the teaching plan.

Based on the OS experiment data gathered over four years, the OS experiment can easily scale to a large number of students without affecting learning effectiveness. Furthermore, the behavior tracking mechanism can also effectively provide feedback during the learning process, which is crucial for the teachers to provide personalized guidance in a short time.

The findings of this article are useful for both the teachers and the students of the OS experiment course. For the teachers of the OS experiment course, the findings show that it would be beneficial to provide step-by-step guidance for the students in an OS experiment course. Furthermore, cloud computing infrastructure, virtualization technology, and an automatic code evaluation system are crucial to improve the scalability of the OS experiment to a large number of students. Finally, learning behaviors, such as login duration, log size, and file open count are all useful features correlated with the performance of the students in the course. The study clearly indicates that it would be beneficial for the students to work on the projects based on the step-by-step guidebook. Moreover, it is also valuable for the students to keep an eye on their learning behaviors, to analyze them, and to improve their learning effectiveness in time, based on the analysis result.

There are two limitations with the current OS experiment platform. First, the current platform is still not deployed on the Internet. Public cloud computing infrastructure and docker techniques may be used to make the experiment platform available on the Internet in the near future. Second, the current experiment platform and its related resources are still in Chinese. Translating the experiment platform and the teaching resources into English would make it useful for a broader audience.

References

[1] Joint Task Force on Computing Curricula, *Computer Science Curricula 2013*. New York, NY, USA: ACM Press, 2013.

[2] *Operating System Engineering*, Massachusetts Inst. Technol., Cambridge, MA, USA, 2019. [Online]. Available: https://pdos.csail.mit.edu/6.828/2018/overview.html

[3] A. S. Tanenbaum and A. S. Woodhull, *Operating Systems: Design and Implementation*, vol. 2. Englewood Cliffs, NJ, USA: Prentice-Hall, 1987.

[4] *The Pintos Project*, Stanford Univ., Stanford, CA, USA, 2009. [Online]. Available: https://web.stanford.edu/class/cs140/projects/pintos/pintos.html

[5] E. T.-H. Chu and C.-W. Fang, "CALEE: A computer-assisted learning system for embedded OS laboratory exercises," *Comput. Educ.*, vol. 84, pp. 36–48, May 2015.

[6] R. Cox, M. F. Kaashoek, and R. Morris. (2019). *XV6, A Simple Unix-Like Teaching Operating System*. [Online]. Available: https://pdos.csail.mit.edu/6.828/2019/xv6.html

[7] R. R. Otero and A. A. Aravind, "MiniOS: An instructional platform for teaching operating systems projects," in *Proc. 46th ACM Technol. Symp. Comput. Sci. Educ.*, Feb. 2015, pp. 430–435.

[8] M. Ziwisky, K. Persohn, and D. Brylow, "A down-to-earth educational operating system for up-in-the-cloud many-core architectures," *ACM Trans. Comput. Educ.*, vol. 13, no. 1, 2013, Art. no. 4.

[9] *The JOS Operating System*, Massachusetts Inst. Technol., Cambridge, MA, USA, 2019. [Online]. Available: https://pdos.csail.mit.edu/6.828/2007/

[10] T. Anderson. (2019). *Not Another Completely Heuristic Operating System*. [Online]. Available: https://homes.cs.washington.edu/~tom/nachos/

[11] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola, "Comparing effective and ineffective behaviors of student programmers," in *Proc. 5th Int. Workshop Comput. Educ. Res.*, 2009, pp. 3–14.

[12] M. Fuchs, M. Heckner, F. Raab, and C. Wolff, "Monitoring students' mobile app coding behavior data analysis based on IDE and browser interaction logs," in *Proc. IEEE Global Eng. Edu. Conf. (EDUCON)*, İstanbul, Turkey, 2014, pp. 892–899.

[13] D. Hou and L. Li, "Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions," in *Proc. IEEE 19th Int. Conf. Program Comprehension (ICPC)*, Kingston, ON, Canada, 2011, pp. 91–100.

[14] T. G. Cleaver and L. M. Elbasyouni, "Student online assessment behaviors," *IEEE Trans. Educ.*, vol. 48, no. 3, pp. 400–401, Aug. 2005.

[15] L.-Y. Li and C.-C. Tsai, "Accessing online learning material: Quantitative behavior patterns and their effects on motivation and learning performance," *Comput. Educ.*, vol. 114, pp. 286–297, Nov. 2017.

[16] *Operating System Experiment Guide Book (in Chinese)*, Beihang Univ., Beijing, China, 2019. [Online]. Available: http://jiangbo.buaa.edu.cn/OSLabGuide.pdf

[17] M. Guzdial, "What's the best way to teach computer science to beginners?" *Commun. ACM*, vol. 58, no. 2, pp. 12–13, 2015, doi: 10.1145/2714488.

[18] R. R. Otero and A. Aravind, "MiniOS: An instructional platform for teaching operating systems labs," *CoRR*, vol. abs/1811.09792, pp. 1–32, Dec. 2018.

[19] X. Chen, L. Breslow, and J. DeBoer, "Analyzing productive learning behaviors for students using immediate corrective feedback in a blended learning environment," *Comput. Educ.*, vol. 117, pp. 59–74, Feb. 2018.

[20] *Architecture Related Questions Used in OS Experiment Labs*, Beihang Univ., Beijing, China, 2019. [Online]. Available: http://jiangbo.buaa.edu.cn/Architecture.pdf

[21] *Architecture Related Lab Tests*, Beihang Univ., Beijing, China, 2019. [Online]. Available: http://jiangbo.buaa.edu.cn/archLabs.pdf

[22] J. J. P. Arias, J. G. Duque, R. D. Redondo, and A. F. Vilas, "COSTE: Open environment for teaching in computer science area," in *Proc. Frontiers Educ. Conf.*, vol. 3. San Juan, Puerto Rico, USA, 1999, Art. no. 13B9/14.

[23] D. M. Hassan, D. Leclet, and B. Talon, "An online laboratory in a context of project-based pedagogy," in *Proc. 9th IEEE Int. Conf. Adv. Learn. Technol.*, Riga, Latvia, 2009, pp. 128–130.

[24] B.-S. Jong, C.-H. Lai, Y.-T. Hsia, T.-W. Lin, and C.-Y. Lu, "Using game-based cooperative learning to improve learning motivation: A study of online game use in an operating systems course," *IEEE Trans. Educ.*, vol. 56, no. 2, pp. 183–190, May 2013.

[25] T. R. Liyanagunawardena, A. A. Adams, and S. A. Williams, "MOOCs: A systematic study of the published literature 2008–2012," *Int. Rev. Res. Open Distrib. Learn.*, vol. 14, no. 3, pp. 202–227, 2013.

[26] N. Lung-Guang, "Decision-making determinants of students participating in MOOCs: Merging the theory of planned behavior and self-regulated learning model," *Comput. Educ.*, vol. 134, pp. 50–62, Jun. 2019.

Lei Wang received the B.Sc. degree in computer science from Shenyang Jianzhu University, Shenyang, China, in 1991, the M.Sc. degree from Harbin Engineering University, Harbin, China, in 1994, and the Ph.D. degree from Beihang University, Beijing, China, in 1998.

He is an Associate Professor with the School of Computer Science and Engineering, Beihang University. His research interests include operating system, complex networks, and compiler.

Ziqi Zhen received the B.S. degree from the School of Computer Science and Engineering, Beihang University Beijing, China, where he is currently pursuing the M.S. degree in computer engineering.

His current research interests include operating system, embedded system, and deep learning acceleration.

Tianyu Wo received the Ph.D. degree from Beihang University, Beijing, China, in 2008.

He is an Associate Professor with Beihang University. He has been authorized more than 30 patents. His research interests include software of distributed system, system virtualization, spatial–temporal data processing, and applications in Internet of Vehicles. He has published over 70 papers in the above areas.

Bo Jiang (Member, IEEE) received the Ph.D. degree from the University of Hong Kong, Hong Kong.

He is an Associate Professor with the School of Computer Science and Engineering, Beihang University, Beijing, China. His research has been reported in leading journals and conferences, such as ASE, FSE, ICWS, QRS, TSC, TRel, the *Journal of Systems and Software*, *Information and Software Technology*, and SPE. His current research interests focus on software testing, debugging, and blockchain technology.

Hailong Sun (Member, IEEE) received the B.S. degree in computer science from Beijing Jiaotong University, Beijing, China, in 2001, and the Ph.D. degree in computer software and theory from Beihang University, Beijing, in 2008.

He is an Associate Professor with Beihang University. His research interests include intelligent software engineering, crowd intelligence/crowdsourcing, and distributed systems.

Dr. Sun is a member of the ACM.

Xiang Long received the B.S. degree in mathematics from Peking University, Beijing, China, in 1985, and the M.S. and Ph.D. degrees in computer science from Beihang University, Beijing, in 1988 and 1994, respectively.

He has been a Professor with Beihang University since 1999. His research interests include parallel and distributed system, computer architecture, embedded system, and multi/many-core-oriented operating system.