Analysis of Students' Behavior in the Process of Operating System Experiments

Lei Wang Chao Gao Tianyu Wo Bin Shi
School of Computer Science and Engineering, Beihang University
Beijing, China
wanglei@buaa.edu.cn, heroism.1993@gmail.com, woty@act.buaa.edu.cn, shibin@act.buaa.edu.cn

***Abstract*—Operating system (OS) experiments consolidate the understanding of the OS concepts and cultivate good engineering practices. Major challenges, however, including large class sizes, diverse software versions, and timely identification of difficulties from lab reports, hurt teaching quality. To address these, we designed an integrated environment to support OS experiments and automated the release and testing of lab code. The environment helped students to write, debug, and run code, and it collected their behavior data. These data included login hours and frequency, commands executed, files opened, and the code submission frequency. We found through analyzing the data a lack of preliminary knowledge in some students. Extra instructions and extended deadlines helped them master the subjects. For some students, the files read indicated the lack of attention to some most relevant system architecture code. This finding allowed us to provide targeted and specific help. Other data revealed an interesting "lab 2" phenomenon where the behavioral difference between the competent and average students was much more pronounced for lab 2 than that for lab 1. The data showed that login hours and frequency and log size correlated positively but submission frequency correlated negatively with grades. Analysis of students' behavior data allowed us to realize continuous improvements in the experiment process. 62% of the 152 students completed four labs and 46% all six, a significant improvement over the last semester.**

Keywords—Student Behavior; Operating System Experiments; Teaching Process

I. INTRODUCTION

An operating system (OS) manages computer hardware and software resources. An OS course not only teaches students theories, but also OS design and implementation techniques. As a vital part of the course, OS experiments consolidate students' understanding of concepts and principles and cultivate good engineering practice. We conduct experiments on various OSes, such as Linux, Windows, etc. Finally, we have introduced MIT OS [1] experiments where students are required to complete six labs and implement a small OS in one term. These labs are designed to greatly improve hands-on capabilities.

In 2014, however, only 38% of a large class (255 students) completed four labs. This completion ratio was lower than expected. Reflecting on the experience, we discovered the following causes:

- A large class size limited the amount of attention for individual students. Since the number of students and the amount of homework have increased, it was difficult for the teacher to provide detailed and timely guidance.
- It could be difficult to spot problems from lab reports in time. The lessons learned could only be passed onto the next semester. For example, when insufficient prerequisite knowledge for OS experiments was discovered at the end the term, the problem could only be solved in next term.
- OS experiments needed Linux, GCC, simulator and other complex software. Without a consistent experiment environment, different versions of the software could cause problems.

As emphasized in software engineering that software quality is embedded in the software development process, **teaching quality is also embedded in the teaching process**. Therefore, the basic idea to solve these problems was to manage the whole teaching process by automating the experiment releasing, submission, judgment and related works. Automation reduced the workload of the instructor. Meanwhile, it is used to help analyzing the students' behavioral data and to provide detailed guidance.

The major contributions of this paper are presented as follows:

- We design our OS experiments based on MIPS. In the course, students are required to implement a small OS in one term. This target is divided into 6 successive labs. Each of them is aimed for part of OS kernel functions, and based of the last lab. In this way, students could get a further understanding of OS.
- We implement an integrated environment to support OS experiments. The tools necessary, such as GCC, simulator, editor, are integrated in the environment. There's no need for students to prepare their own environment. In this environment, we use *git* for source control, and automate the release and test the code based on *git* hooks. At the same time the environment collects data about students' behavior in the process of OS experiments.
- We have found through analyzing students' behavior data the lack of preliminary knowledge and the lack of understanding of experiment code in some students. In addition, the data revealed an interesting "lab 2"

This work was supported by National Natural Science Foundation of China (No.61272167) and the 863 Program (2015AA01A202).

phenomenon where the difference of behavior data between good and average students was pronounced for lab 2 while all students' behavior data was almost the same for lab 1. Correlation analysis of the behavior data and grades revealed login hours, frequency and log size correlated positively with grades. There is a negative one between submission frequency and grades.

The paper is organized as follows: Section II gives a short overview of related work. We introduce our MIPS-based OS experiments in Section III and describe the integrated environment in Section VI. During the experiment process, all steps, including automatic code releasing, coding, compiling and running, code submission, automatic testing and evaluation results feedback, are supported by the integrated environment. The environment avoids software installation and version discrepancies. Additionally, the environment collects students' behavioral data, including the login hours, the login frequency, the commands executed, the files opened, and the code submission frequency. We analyze these data and their correlation with grades in Section V. The paper closes with our conclusions in Section VI.

II. RELATED WORK

Operating systems has become part of the computer science core curriculum since early 1970s [2]. Existing approaches of teaching operating systems courses and designing experiments try to enhance the effectiveness and efficiency of learning by introducing online course systems and automated teaching and assessment tools.

A. Lab design of operating systems courses

One of the most popular syllabus designed for operating systems lab practice was introduced by Tanenbaum [3]. Students may learn operating system principles as well as implementation techniques with the MINIX operating system. Code reviewing of MINIX used to be a lab practice activity. Other educational operating systems are developed such as Nachos [4], Pintos [5] and Xv6 [6]. They are used to prompt the understanding of operating system concepts by requiring students to implement some important functions. Most of these operating systems run in simulators so that students can easily develop and debug their codes. Some off-the-shelf operating systems such as Linux and Windows are used in lab projects as well. In these cases codes are developed by students to implement important algorithms in operating systems such as process scheduling and memory allocation. We believe that the best way of mastering operating system principles is to design and implement one by oneself. Thus, based on JOS [1], we also introduce the lab projects requiring students to fill out some important code in the core functions. In our lab design, an operating system running on MIPS architecture is provided. Students can run it using a simulator such as GXEMUL [7]. One of the differences between our approach and above works is that we divided the whole lab course into 6 successive projects and students are fed with a new project only if they finish the previous one. By using this method, we can know about the learning progress of each student and make them better focusing on the working feature.

B. Tools and platforms

Internet brought up a brand new paradigm of teaching and learning. Some early experience of online course practices for computer science took place in 1990s. Pazos Arias *et. al.* proposed COSTE [8] as a web based online course system. It provided course material and online studying and exam functions. Hassan *et. al.* [9] proposed an online laboratory for project-based pedagogy by using virtualization technologies. Stylos *et. al.* [10] studied the programmers' web search behavior by using a specific software interfaces. Mamykina *et. al.* [11] analyzed a Question & Answer site for software developers. Students' programming behavior has been analyzed using various types of data, including analysis of graded programming assignments [12], browser Interaction logs [13] or students' newsgroup discussions [14]. Massive Open Online Courses (MOOCs) emerged in 2008 and are widely adopted these years [15]. Some platforms are developed such as Edx, Coursera and Udacity. They provide fine granularity of knowledge transferring, individualization of learning process and data based analysis. While these platforms emphasis on feeding students with split video clips, online testing and interactive discussion zones, the lab practice courses can be hardly supported. Our platform amended this by providing an environment of system software programming. A fine granularity learning activity monitor which can gather student behavior in detail is deployed. It can be seen as a new extension to the MOOCs paradigm of lab practicing curriculums. The automation exists not only in the course assessment and learning progress control, but also in the data driven learning effectiveness evaluation.

C. Platform for other CS courses

Online judge systems are widely used in programming contests as well as introduction to programming courses (CS1) and basic data structures courses (CS2) [16]. These systems provide students an interface to submit their programs, they compile and run test cases against the students' code automatically and score the programs according to the test result. By using these tools, teachers can check students' homework and evaluate their programing ability even for a large class. And the student can get real-time feedback without waiting for next class or going to the TA's office. As the first two basic courses for computer science major, CS1 and CS2 only require students to master the fundamental principle and programming techniques. The programs that written by students are relative short. Normally it only consists of several functions with less than two hundred lines of code. The dependencies of the code are also limited. So the online judge systems only have to provide interfaces such as a text input box or a single source code file submission function. Operating

systems, on the other hand, are much more complex. Source code for the simplest OS can have several thousand lines of code. It is not easy for students to write an OS from scratch. It's even harder to judge the correctness of OS functions since they may have mutual dependencies and the function interfaces can be different. So in our lab design, a full development environment, rather than a simple text editor, is provided. Students have to build their code and submit the whole project for judgment. These are the real jobs an OS writer does every day. We do share the same objective of making faster feedback to students as systems used in CS1 or CS2. Meanwhile we have further target to help students to master the procedure of developing a big system software with multiple source files.

III. A BRIEF INTRODUCTION OF THE OS EXPERIMENTS

In the course, we ask students to develop a small OS on the MIPS platform [17]. The experiments are split into six major labs that build on each other, culminating in a primitive operating system on which students can run simple commands through their own shell. The details are:

(1) Boot and System Initialization: To analyze the hardware boot process to understand OS kernel linking, loading and relocation, and to implement a printf.

(2) Memory Management: To understand the MIPS memory layout, and implement physical and virtual memory management.

(3) Process Management: To implement clock interrupt handler, process creation, termination, scheduling, and management schemes.

(4) System Call: To understand the system call mechanism on MIPS and implement system calls.

(5) File System: To implement a simple file system.

(6) Shell: To implement a basic shell and combine the six parts to form a small OS.

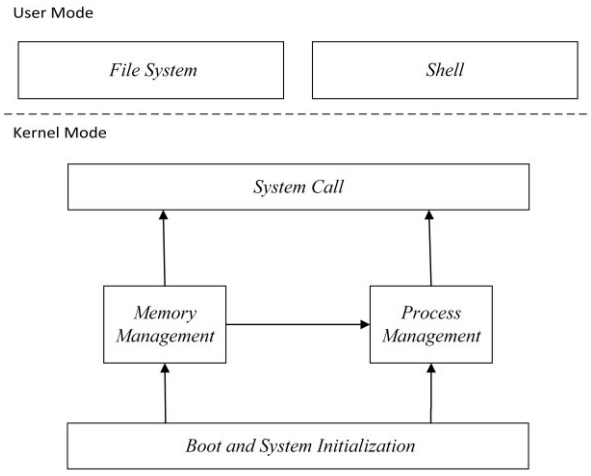

Fig. 1. Relationships of the six labs

The relationship of the six labs is shown in Fig. 1. In Fig. 1, the six labs are divided into user mode and kernel mode. The labs in the user mode are dependent on those labs in kernel mode and the following lab is dependent on the previous lab. Labs are also getting more and more difficult.

IV. THE INTEGRATED ENVIRONMENT OF OS EXPERIMENTS

Our integrated environment supports the whole process of the experiment course, including the initial code releasing, coding, compiling and running, code submission, testing, and evaluation results feedback. To collect and analyze students' behavioral data in the process, we have designed a tracking module in the integrated environment. We have adopted git [18] for code version control. The students are doing experiments under supervision. By using git hooks and modified ssh server, any command line input and source code submissions are recorded. The density of interaction between students and the system can also be used to infer the student's efforts in OS studying. The integrated environment controls the progress of students' OS programming practices and collecting data from any individual student. We can review the progress of a student finishing those 6 labs. The correlation between students' behavior and their performance can be used to improve the lab design and teaching methods.

The structure of the environment is shown in Fig. 2. It consists of the following parts:

- The virtual machine platform which includes the development platform and tools, e.g., Linux, the cross compiler toolchain, and the MISP simulator etc.
- The git server, which includes the initial code for lab and the code from students.
- The automatic evaluation and feedback module, which is integrated in the git server. It tests lab codes submitted by students automatically. The results are fed back to the students through the git server.
- The learning process tracking module, which is integrated in the virtual machine and collects the students' behavioral data in the learning process.

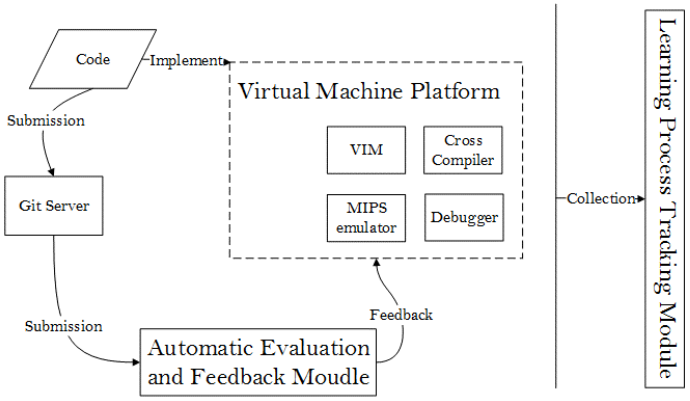

Fig. 2. Integrated Environment Structure of OS Experiments

A. *The Virtual Machine Platform*

We provide a virtual machine as the experiment platform where a Linux system is installed and the Gxemul simulator, the gcc cross-compiler for MIPS, the vi editor and other software are deployed. Fixing the versions of standard tools avoids discrepancy and lays a foundation for the automatic evaluation system. It saves the time for students setting up their own environments. Each student has a Linux account of general authority in the virtual machine platform. The students can log in our system only with a ssh client to complete the labs. Meanwhile, the system collects behavioral data in the learning process through these accounts.

B. *Git Server*

To manage code release and submission, we provided a git server. Code is written and tested by the students in the virtual machine platform and collocated in the git server. The lab code release and test results feedback are also completed via the git server. In the git server, each student creates his/her own branches from code library and submit their branches to the codes library. All branches of code libraries are visible to teaching assistants and the teacher.

C. *The Automatic Evaluation and Feedback Module*

The number of students is large and code review takes huge effort. We have implemented an automatic evaluation module which scores students' code automatically. When code is submitted via the git server, the evaluation module is triggered. It tries to compile the code, run and test the program, and gives a score. The results are fed back to the students through the git server. Once a student's code passes the evaluation, the code of next lab is released to him or her. In this way, more competent students are encouraged to finish more labs.

D. *The Learning Process Tracking Module*

It is easier to discover problems if we could track the students' learning process. Without understanding the process, it is hard to help or give detailed guidance. In the integrated environment, we have implemented a learning process tracking module. We can collect students' behavioral data in the experiment process through ssh remote access, shell historical records and git server records. The learning pattern and existing problems can be revealed by analyzing the behavioral data.

E. *The Results*

In 2015, the integrated environment for OS experiments was completed and put to work. Each student could log in his/her own account, check out his/her own branches of code library and start coding (as Fig. 3). They could create their own branches for version control. When a lab was complete, the code was pushed to the git server (as Fig. 4) and evaluation was triggered automatically (as Fig. 5). If the code passed the evaluation, the next lab code would be pushed to the code library of the student.

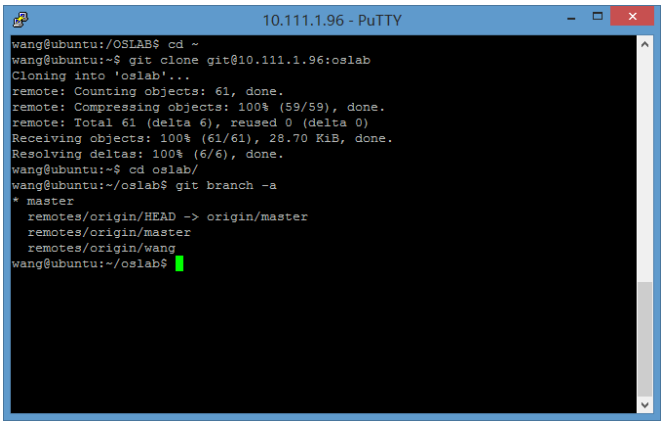

Fig. 3. Login and Codes Branch Creation

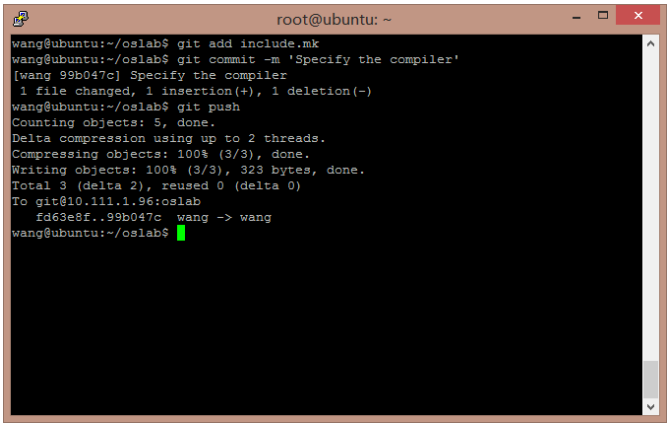

Fig. 4. Code submission

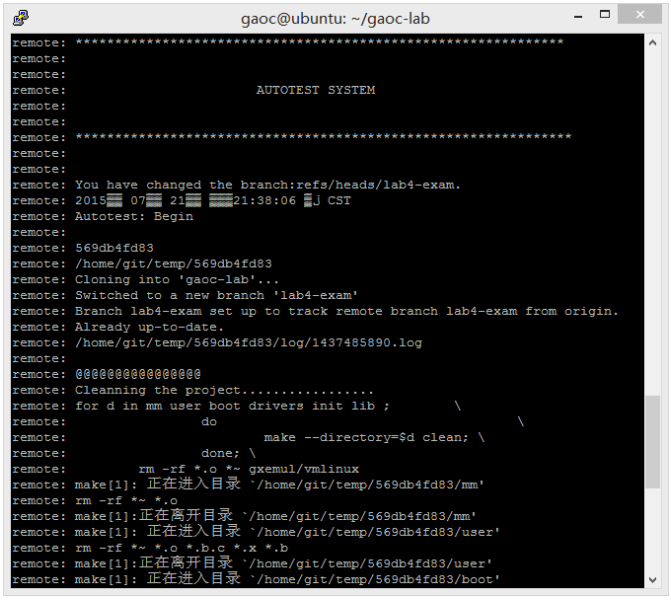

Fig. 5. Automatic Evaluation

V. Behavioral Data Analysis

A. Data Collection

The students are notified in the first class that the system is collecting all their behavioral data on the platform. Although the user identity can be acquired from the login name, the analysis results do not contain any judgment of individual student. The privacy of students' behavioral data is preserved. Behavioral data of students are collected from three sources in the integrated environment:

- *The ssh sessions*. Since ssh remote shell is the only way to access the experiment environment, we can capture system time, login account name, user input information stored in the buffer, etc.
- *The historical records of shell commands*. These records can provide information on commands executed, files opened, etc.
- *The records of git server*. The code library operation data on the git server.

1) Data from ssh Sessions

From ssh sessions, we have collected the students' login information. The time stamps in the login file record the login frequency and duration. Thus, the information of login hours and login frequency of the student is obtained. Fig. 6 shows the daily total login hours of all students in lab1. Fig. 7 shows the daily login frequency sum of all students in lab1. The x-axes in Fig. 6 and Fig. 7 represent the days in lab1, and the y-axes are seconds and times of login respectively. From Fig. 6 and Fig. 7, as the increasing of time, both the login hours and login frequency tend to decrease.

Fig. 8 shows the login distribution from between 0:00 and 23:00 in a day. From Fig. 8, one can see that most students login the system and start experiments at 18:00. Table I shows the mean login frequency and login hours in four labs. Because only 46% students completed lab5 and lab6, we collected these data of four labs in Table I. It can reflect to some degree the lab's difficulty and the students' efforts.

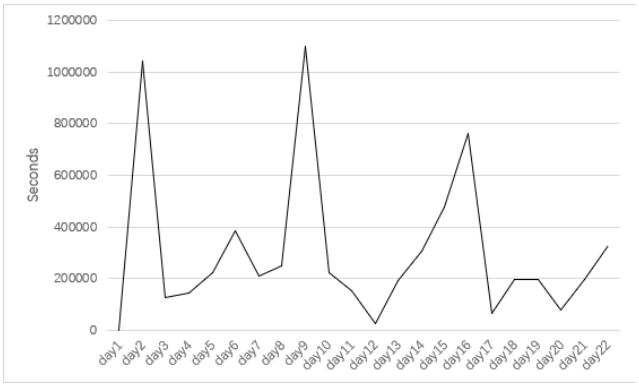

Fig. 6. Daily Login Hours Sum of lab1

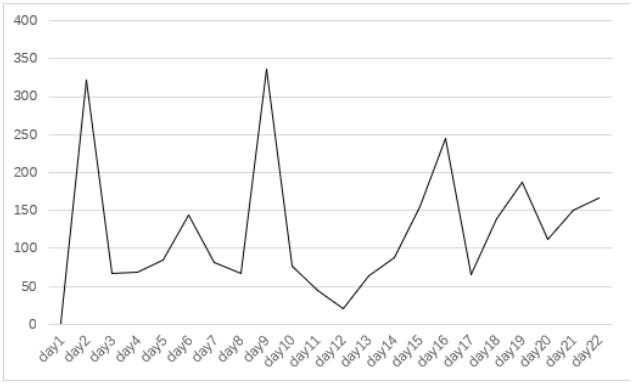

Fig. 7. Daily Login Frequency Sum of lab1

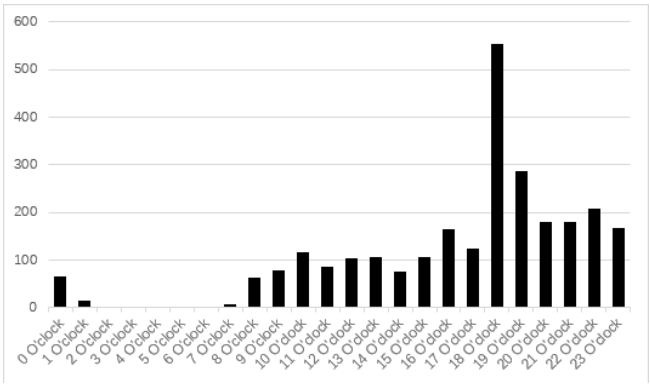

Fig. 8. Login Time Distribution of lab1Students in a day

TABLE I. Mean Login Frequency and Login Hours of Students

	Mean Login Frequency	***Mean Login Hours***
Lab1	21.862 times	43714 s
Lab2	17.618 times	35146 s
Lab3	10.784 times	26183 s
Lab4	24.209 times	28648 s

2) Historical Records of shell Commands

The historical records of shell commands provide commands executed and files opened in the experiment. Fig. 9 shows the most frequently used 9 commands in lab1, i.e. cd, ls, vi, vim, make, git, gcc, source and cp. The x-axis in Fig. 9 represents the days in lab1, and the y-axis is frequency of a command. From Fig. 9 we found a preliminary knowledge problem (the details can be found in the next section).

Similarly, Fig. 10 shows the 20 files opened most in the whole experiment. The x-axis in Fig. 10 is the file names, and the y-axis represents the times a file is opened.

3) Collected Data from the git server

Much data can be collected on the git server. Currently, we are most interested in code submission frequency. Git keeps version information locally before submission. When

the code is submitted, it is synchronized with the remote repository. We have made statistics on daily code submission during the experiment process for all students. Fig. 11 shows the code submission frequency in lab 1. The x-axis in Fig. 11 represents the days in lab1, and the y-axis is times of the code submission.

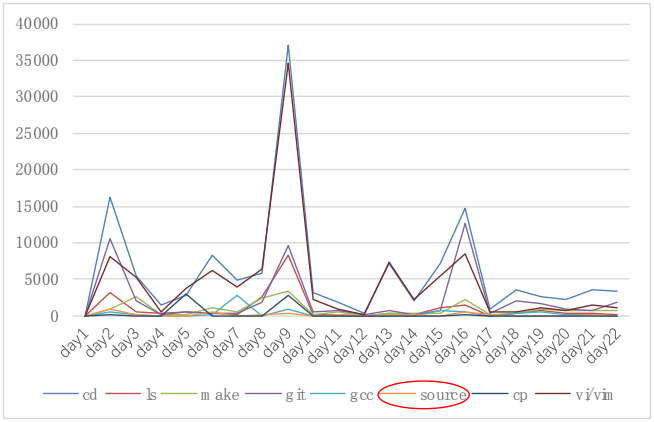

Fig. 9. Used Commands of students in lab1

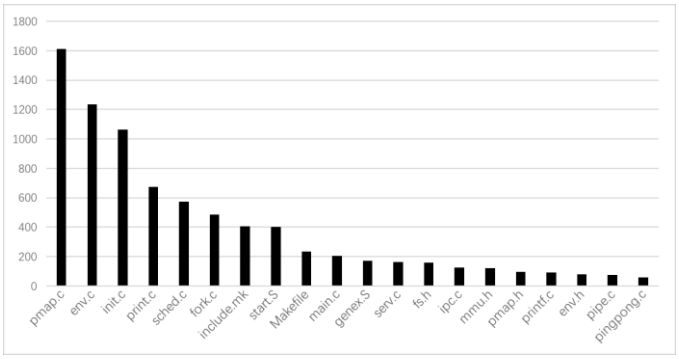

Fig. 10. File Opening Times

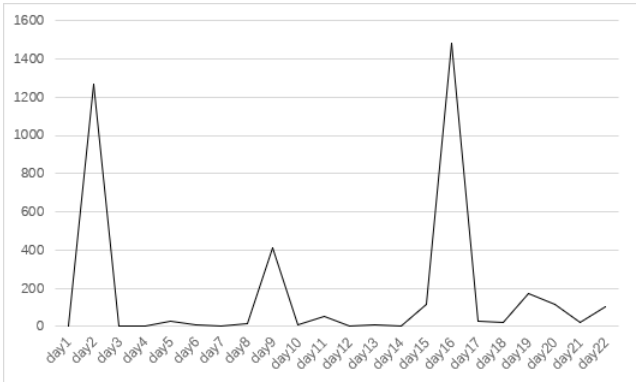

Fig. 11. Codes Submission Information

B. *Problems Discovered from Behavioral Data*

Through the behavioral data collected in the experiment-integrated environment, we have discovered problems timely. These problems include the lack of preliminary knowledge, the lack of understanding of lab code, and the "lab2" phenomenon.

1) The Lack of Preliminary Knowledge

By analyzing the command frequency data during 22 days of lab1, we discovered that the source command (command in the red circle of Fig. 9) was used frequently and for a long time. This was abnormal, because the source command was typically for system configuration. The high frequency indicated that some students were not familiar with the system. Therefore, we collected polls on 29 students and investigated the degree of familiarity with Linux, GCC, make, vi, Gxemul, lds, shell and git. The results were shown in Table II. The data indicated that the students were unfamiliar with the basic tools used in the OS experiment. This affected the experiment schedule. To deal with this problem, we increased the duration of lab2 from 2 to 3 weeks in the hope that students have more time to learn about the basic tools to be able to keep up with the labs.

TABLE II. FAMILIARITY DEGREE OF THE BASIC TOOLS

Tools	***familiarity***	***used***	***unused but known***	***unknown***
Linux	3.45%	55.17%	31.03%	10.35%
GCC	0%	55.17%	27.59%	17.24%
make	0%	44.83%	37.93%	17.24%
vi	17.24%	58.82%	10.35%	13.79%
Gxemul	0%	27.59%	34.48%	37.93%
lds	0%	31.03%	17.24%	51.73%
shell	10.35%	48.27%	20.69%	20.69%
git	10.35%	62.07%	13.79%	13.79%

2) The Lack of Understanding of Experiment Code

To reduce the OS experiment difficulty, we provided some architecture-related code (in assembly) to the students in the hope that the code could be examined for a better understanding of OS. By analyzing the files opened (as in Fig. 10), however, we found that only two assembly files were among the often opened files. Namely, most students did not read the architecture-related code, and very likely did not understand the OS design at a detailed level. In response, we further explained these code in this term. In next term, we plan to add more questions on these code to give the students a better understanding.

3) The lab2 Phenomenon

Students who completed labs 5 and 6 achieved a better understanding of OS. On login frequency, login hours, code submission frequency and file opening times, we compare the data for those students with the mean values of the grade. Table III revealed an interesting “lab 2” phenomenon.

In lab 1, there was little difference between the data for those students who completed labs 5 and 6 and the mean values. In lab 2, the login hours and login frequency for these high achievers were 21% and 45% higher than the mean, respectively, while the git submission frequency for them was 22% lower than the mean. Lab 2 data showed that the students who had better performance spent more time and energy on labs. Additionally, the average number of files opened by these students was 147.4, 46% higher than the mean value for the grade, 101.2. This indicated that these students spent much more time to read OS code. This phenomenon also indicated that more students were motivated in lab 1 than in lab 2. How to keep the motivation is a problem to be solved in the future.

C. *Correlation Analysis of Students' Behavioral Data and Grades*

We have analyzed the correlation between the behavioral data and the final grade. These analyses may help us to understand the causes affecting experiment course results and to improve the experiment process continuously.

TABLE III. COMPARISON BETWEEN STUDENTS WITH EXCELLENT ACHIEVEMENTS AND MEAN VALUE

	Lab1		***Lab2***	
	Excellent	***Mean***	***Excellent***	***Mean***
Login Hours	44277.4s	43714.1s	42565.3s	35146.8s
Login Frequency	22.8times	21.9 times	25.5 times	17.6 times
Submission Frequency	25.4 times	25.4 times	59.5 times	76.7 times

Firstly, we correlated login hours, login frequency, code submission frequency, the size of the log recorded in the experiment process with the grade of each student. We have eliminated five groups of abnormal data of code submission frequency (the submission frequencies over 500). This might be caused by unfamiliarity of students to git. The results are shown in Table IV which revealed a positive relationship between login hours and grades, a positive relationship between login frequency and grades, a slightly negative one between submission frequency and grades, and a positive relationship between size of the log recorded and grades.

TABLE IV. CORRELATION BETWEEN STUDENTS' BEHAVIORAL DATA AND THE GRADES

	correlation coefficient R
Login Hours	0.526
Login Frequency	0.595
Submission Times	-0.065
Size of the log recorded	0.620

Form the above analysis, the correlation between the size of the log recorded and grade is stronger than those between the login hours and grades and login frequency and grades. Some students might be simply idling although they did log in frequently and spent more time online. The size of the log recorded reflected the activity of a student after login and therefore may have a stronger correlation with the grades.

TABLE V. GRADE SEGMENT AND SIZE OF THE LOG RECORDED

	Mean size of the log recorded
90+ Students	880.1240594MB
Normal Students	316.1920599MB
60- Students	25.97135162MB

Since the size of the log recorded can better predict learning results, we take further analysis for the log size. Table V shows the sizes of the logs recorded of students at different grade levels. Viewing from Table V, the mean size of the log recorded for the students who earned over 90 scores was 880MB and that for all students was 316MB. The mean size of the log recorded of students below 60 was only 25.9MB.

Fig. 12 showed the ratio of different size of the log recorded in different grades. Fig. 12 showed the grades of students with 1000MB+ logs all earned above 90 scores. 13.64% of the students with 600M~1000MB logs earned above 90 scores, 81.82% earned 75-90 scores, and 4.54% earned 60~75 scores. Among the students with 200~600MB logs, 51.22% students earned 75~90 scores and 48.78% students 60~75. Among the students with less than 200MB size of the log recorded, 22.86% students got 75-90, 34.29% got 60~75 and 42.85% got below 60 scores. Some of the students with less than 200MB logs, however, still earned good grades. Upon further investigation we discovered that they downloaded and wrote codes on their own machines and just submitted the codes in the integrated environment. This explained the smaller logs.

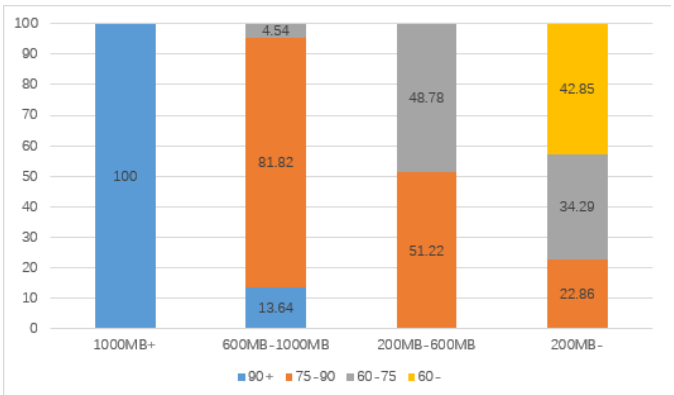

Fig. 12. Achievements Distribution of Different Log Amounts

VI. Conclusion

To improve OS experiments, we have designed an integrated environment, and implemented automatic releasing and testing of students' experiment code. The integrated environment supports coding, compiling and program execution. Meanwhile, we have collected behavioral data including login hours, login frequency, input commands, opened files, code submission frequency and logs recorded by the environment.

By timely analyzing the behavioral data during lab1, we discovered problems including insufficient preliminary knowledge and lack of attention to hardware architecture-related codes. In lab2, we have solved these problems by allowing more time and extending explanations for hardware-related codes. We have discovered an interesting "lab 2" phenomenon. Starting at lab 2, behavioral data of students with better grades showed obvious deviation from the mean, but those in lab 1 had little difference.

From the behavioral data, we have discovered that the login hours and frequency have a positive correlation, and the code submission frequency has a negative correlation with the grades. We have also discovered that the size of the logs is also in positive correlation with the grades.

By monitoring the OS experiment environment, we can discover and solve problems more timely. We have implemented detailed guidance for students and made continuous improvements for the experiment process. In 2014, without the integrated experiment environment, only 38% of students completed four labs and 8% completed lab5 and lab6. In 2015, using integrated environment, 62% students completed four labs and 46% completed lab5 and lab6. From these data, the integrated experiment environment has greatly improved the teaching quality.

The OS lab courses can be improved by taking the students' behavior into consideration. We plan to adjust the time limit for each lab, and making the learning curve smoother by giving more time to the difficult labs. Some competition mechanisms can be introduced to improve the student participation. The detailed analysis of individual labs and students activities can be realized after we collected more data in the future years.

It is obvious that OS experiments and the integrated environment are mutually independent. The virtual machine platform, the git server, the automatic evaluation module and the learning process tracking module can be easily applied to other courses such as Computer Science 1 (CS1) and Computer Science 2 (CS2). And we plan to add a programming course in the integrated environment next term. We also plan to improve the system by introducing more user-friendly tools besides the powerful vi. Providing graphical UIs for students doing the experiments is also under consideration. The code instrumentation and auto testing function for interactive tasks such as lab 6 (shell) need to be improved.

Acknowledgment

We gratefully acknowledge the valuable comments of Dr. Xiaopeng Gao and the valuable data of Qian Liu.

References

[1] MIT 6.828: Operating System Engineering/Fall 2009 http://pdos.csail.mit.edu/6.828/2009/schedule.html, December 2009.

[2] P. J. Denning, Fifty Years of Operating Systems. Communications of the ACM, 59(3), p.30-32, March 2016.

[3] A. S. Tanenbaum, A. S. Woodhull, Operating Systems: Design and Implementation, 3rd Edition, Prentice Hall, 2006.

[4] Thomas Anderson, Not Another Completely Heuristic Operating System, https://homes.cs.washington.edu/~tom/nachos/, 1992.

[5] Ben Pfaff, Pintos Projects, http://www.stanford.edu/class/cs140/projects/pintos/pintos.html, 2004

[6] MIT Xv6, a simple Unix-like teaching operating system, https://pdos.csail.mit.edu/6.828/2011/xv6.html, 2006

[7] The GXEMUL homepage, http://gxemul.sourceforge.net/, 2016

[8] J. J. Pazos Arias, J. Garcia Duque, R. Diaz Redondo, A. Fernandez Vilas, "COSTE: Open Environment for Teaching in Computer Science Area," Frontiers in Education Conference, vol 96, 1999.

[9] T. R. Liyanagunawardena, A. A. Adams, S. A. Williams, "MOOCs: A Systematic Study of the Published Literature 2008-2012" The International Review of Research in Open and Distance Learning, Vol 14, No 3, pp.202-227, 2013.

[10] J. Stylos, B. A. Myers, "Mica: A Web-Search Tool for Finding API Components and Examples," In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, pp.195-202, 2006.

[11] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, B. Hartmann, "Design lessons from the fastest q&a site in the west," In the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vancouver, BC, Canada, pp.2857-2866, 2011.

[12] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, B. Tretola, "Comparing effective and ineffective behaviors of student programmers," In the Proceedings of the fifth international workshop on Computing education research workshop, ACM, Berkeley, CA, USA, pp.3-14, 2009.

[13] M. Fuchs, M. Heckner, F. Raab and C. Wolff, "Monitoring students' mobile app coding behavior data analysis based on IDE and browser interaction logs," 2014 IEEE Global Engineering Education Conference (EDUCON), Istanbul, pp. 892-899, 2014.

[14] D. Hou, L. Li, "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions," In the Proceedings of IEEE 19th International Conference on Program Comprehension, p.91-100, 2011.

[15] Djama M. Hassan, Dominique Leclet, Bénédicte Talon, "An Online Laboratory in a Context of Project-Based Pedagogy," Ninth IEEE International Conference on Advanced Learning Technologies, pp.128-130, 2009.

[16] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, Otto Seppälä, "Review of recent systems for automatic assessment of programming assignments". Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Pages 86-93, ACM New York, NY, USA 2010

[17] Lei Wang, "The design of operating system experiments," Computer education, 2009(17), pp.16.

[18] Git.git –distributed-is-the-new-centralized. http://git-scm.com/doc