Teaching Operating Systems Using Virtual Appliances and Distributed Version Control

Oren Laadan
Dept of Computer Science
Columbia University
New York, NY 10027
orenl@cs.columbia.edu

Jason Nieh
Dept of Computer Science
Columbia University
New York, NY 10027
nieh@cs.columbia.edu

Nicolas Viennot
Dept of Computer Science
Columbia University
New York, NY 10027
nviennot@cs.columbia.edu

ABSTRACT

Students learn more through hands-on project experience for computer science courses such as operating systems, but providing the infrastructure support for a large class to learn by doing can be hard. To address this issue, we introduce a new approach to managing and grading operating system homework assignments based on virtual appliances, a distributed version control system, and live demonstrations. Our solution is easy to deploy and use with students' personal computers, and obviates the need to provide a computer laboratory for teaching purposes. It supports the most demanding course projects, such as those that involve operating system kernel development, and can be used by both on-campus and remote distance learning students even with intermittent network connectivity. Our experiences deploying and using this solution to teach operating systems at Columbia University show that it is easier to use, more flexible, and more pedagogically effective than other approaches.

Categories and Subject Descriptors

D.4.0 [**Operating Systems**]: General; K.3.1 [**Computers and Education**]: Computer Uses in Education—*distance learning*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Design, Experimentation, Human Factors

Keywords

Operating systems, computer science education, virtualization, virtual machines, open-source software, version control

1. INTRODUCTION

Programming projects are an important aspect of learning about many computer science subjects, especially operating systems. Hands-on experience with operating system code development, debugging, and testing is crucial for helping students to understand how operating system concepts really work in practice. Many approaches have been developed for providing such programming experience, including systems programming projects, operating system simulation environments, pedagogical operating systems, and kernel development in commercial operating systems. All of these approaches share a common need. Students need to be provided with some common software code base to start from to do their programming projects, and they need to be provided with a consistent computing environment with the right versions of development tools so that their programming projects can be accomplished successfully and evaluated correctly.

Many universities provide computer laboratory facilities to support these computing needs. Instructors work with computer support staff to ensure that the facilities provide the necessary development tools, access to required software code bases, and the correct versions of everything to make all the software work together correctly. These facilities typically provide reliable storage space that is backed up, minimizing the impact of hardware failures on students' work in these facilities. Virtualization software is increasingly being made available on these shared computer facilities to support kernel-level programming projects for operating system courses. A student can be given root and dedicated access to a virtual machine so that the kernel development cycle of plan-implement-reboot-test-debug can be done without inconveniencing other users of the laboratory facilities.

Unfortunately, providing and maintaining these computer laboratory facilities for teaching purposes is often difficult. If facilities do not yet exist, just finding the space for such a laboratory can be hard at many sites, especially those in urban environments with more constrained space availability. This can be ameliorated in part by using space-efficient racks of remotely accessible servers in lieu of laboratories where students can sit, but this requires machine room space with suitable power and cooling which can also be quite limited in availability. Even if computer laboratory facilities already exist, they require computer support staff to maintain the facilities, upgrade and repair machines, and install and maintain software required by instructors for teaching purposes. Because computer support staff can be expensive, they are often in limited supply in a university setting, making it more difficult to complete these tasks in a timely fashion. Computer hardware becomes quickly out of date, and finding funds to periodically upgrade computer laboratory facilities, especially those used primarily for teaching

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

purposes, can be problematic. As a result, such facilities are often underprovisioned for the number of students that need to use the hardware, resulting in overloaded machines especially just before homework assignments are due.

These problems are only exacerbated when it comes to computing support for teaching operating systems. Virtual machines are often used in this context, along with large software code bases such as that for an entire commodity operating system such as Linux. Running virtual machines and frequently building entire operating system kernel trees can tax machine resources much more than running regular applications. They impose higher CPU and memory requirements on already overloaded machines. Students may frequently snapshot their virtual machines so that they can easily revert to an uncorrupted system when a kernel crash takes out the system. While virtual machines already incur high disk space requirements, frequent snapshots by many students can easily use up all available disk space, bringing everything to a grinding halt at the most inopportune times when homework assignments are due. Since students doing operating system kernel projects need root privileges inside the virtual machines and can install anything they want, there are increased security risks associated with the virtual machines. To minimize these risks, computer support staff often limit network access to the virtual machines, which can make it more difficult for students to use them to complete their homework assignments.

A benefit of providing shared computer laboratory facilities for running virtual machines is that both students and instructional staff can access them. This can make grading programming projects easier as instructional staff can just log on to the respective students' virtual machines to evaluate their projects. However, this has its own problems as instructional staff need to schedule time with students when their virtual machines are unavailable, and instructional staff must face the risks of using untrusted virtual machines managed by students that may have keyboard loggers and other malicious software installed inside them.

To address these problems, we observe and leverage three recent trends in computing. First, laptop computers have become inexpensive enough and indispensable for education that all computer science students own one. Second, laptop computers have become powerful enough that they can run almost any software that is needed for gaining hands-on experience in programming projects for courses. Third, virtualization technology has become ubiquitously available at little if any cost to students. For example, VMware has products that are freely available to everyone to run virtual machines, and has agreements with many universities that make all products freely available for academic purposes.

Building on these trends, we introduce a new approach to teaching operating systems based on virtual appliances, a distributed version control system, and live demonstrations. We create a virtual appliance for doing operating system homework assignments which can be easily deployed and run on students' personal computers in virtual machines without interfering with any existing software on the their computers. We combine virtual appliances with a distributed version control system to provide reliable storage for students' homework assignments, support students working together on group homework assignments, and manage the submission and grading of homework assignments. We leverage virtual appliances and the version control system to enable students to do live demonstrations of their work as part of grading their assignments to simplify grading, providing better feedback to students and greater interaction between instructional staff and students to facilitate learning.

Our solution is easy to deploy and use with students' personal computers, and obviates the need to provide a computer laboratory for teaching purposes. It supports the most demanding course projects, such as those that involve operating system kernel development, and can be used by both on-campus and remote distance learning students even with intermittent network connectivity. Our experiences deploying and using this solution to teach operating systems at Columbia University show that it is easier to use, more flexible, and more pedagogically effective than other approaches.

2. VIRTUAL APPLIANCES

Students already have their own computers, but they come in diverse software and hardware configurations. Linux serves as an excellent basis for teaching courses such as operating systems, but most students do not run Linux on their computers. Students would be loathed to install Linux natively on their computers just to take a course, given the risks of adversely affecting their existing software configurations and other already installed applications. Instructional staff would face almost insurmountable hurdles to get Linux installed on all students' computers given the diversity of software and hardware configurations in use.

Virtualization solves this problem by making it easy to run Linux operating system instances in virtual machines without affecting existing software already installed on a student's computer. VMware provides commercial virtualization software products on all major operating systems, and their products are as easy to install and use as other regular applications. We use VMware Workstation 6.5 for Windows, VMware Workstation 6.5 for Linux, and VMware Fusion 2.0 for Mac since these products provide excellent support for software development and are freely available to our students through academic licensing agreements.

Using VMware allows us to introduce virtual appliances to provide all the necessary software we need for teaching operating systems. A virtual appliance is a pre-built software bundle that can be downloaded and run locally inside a virtual machine. Unlike generic virtual machines, a virtual appliance need not be installed and configured. It is readily deployable—students just download and use it, avoiding any installation issues. Instructional staff create a Linux virtual appliance for doing operating system programming projects, with the operating system, development tools, version control software, and all other necessary software already configured and bundled together. Once it is working, the files representing the virtual appliance are turned into a compressed archive and made available for download from the class Web site.

To use the virtual appliance, students install VMware Workstation for Windows or Linux, or VMware Fusion for Mac. Students then download the virtual appliance, uncompress the archive, and just launch it in VMware to start running the appliance. VMware virtualization software ensures that the virtual appliance is completely isolated from the rest of the student's computer, and enables the appliance to run a completely different operating system while coexisting with the one already installed on the student's computer. Since students all use the same virtual appliance, everyone

is ensured of running the exact same environment, avoiding discrepancies that can arise when a program is developed and tested with different compilers and operating system versions. Students are free to change the software and hardware configuration of their virtual appliances. They can always get a clean, unmodified version of the appliance by just downloading it again.

Virtual appliances provide a number of important benefits. First, since they run on students' computers, they obviate the need to provide computer laboratory facilities for teaching purposes, and avoid all the related difficulties. There is no need for the associated hardware costs and computer support staff costs for running such a lab. There are no more concerns about network security for a lab. Second, they provide complete isolation from other software running on the students' computers, ensuring that courseware does not interfere with other software and allowing courseware developed for just one operating system to be used portably across many different operating systems and software configurations without additional development cost. Third, since virtual appliances are pre-configured by the instructional staff, they avoid the installation problems and complications that can be associated with directly installing non-commercial grade courseware on students' computers. Fourth, since students control the hardware that runs the virtual appliance, students have the power and flexibility to allocate resources to the virtual appliance as they see fit. They can allocate more CPU or memory, run more copies of the virtual appliance, and enable full network access to make it easier to access network resources. Finally, since virtual appliances are run locally on students' computers, they make it easy to support remote distance learning students as well as on-campus students. Distance learning students do not need to access a campus computing facilities to work, and can work in their virtual appliances even in the absence of network connectivity.

To teach operating systems, we assigned various programming homework assignments that involved modifications to the Linux kernel. This approach provided the advantages of gaining hands-on experience doing kernel-level software development, learning from examples in a real commercial operating system and how to manage the complexity of a large system, and acquiring skill with a widely used commercial operating system that can be used for future employment. For students to do these homework assignments, we configured a Linux virtual appliance with a small footprint on disk and that boots quickly. The small footprint allows it to be quickly downloaded and consume only a modest amount of disk space. Quick downloads make it easier to retrieve clean copies of the virtual appliance to start over if needed. The fast boot allows faster kernel testing and debugging.

We used the Gentoo Linux distribution, which can be automatically optimized and customized to provide the desired small footprint and fast boot times. We customized our installation for kernel testing and debugging. For this purpose, we do not need all the services typically provided by a full Linux distribution and installed a bare minimum base system. For example, we did not install mail, the cron server, syslog daemon, HAL, D-Bus and ALSA utilities, Bluetooth or WiFi capabilities, or ACPI utilities. Since Gentoo is a source-based distribution, it already contains the toolchain for Linux kernel development. A DHCP client, text editors, and Git, for version control, were installed on top of the base system. We did not need a graphical interface for the virtual appliance for our purposes, though we could install one if desired. Once the system was installed and configured in a virtual machine, we cleaned up the system by removing downloaded sources and removing SSH server keys. To make the appliance available for download, we shrank the disk file by wiping the free space on the drive (`dd if=/dev/zero of=tmp; sync; rm tmp`) and shrinking the disk with the vmware-vdiskmanager command. The size of the bzip2 compressed virtual appliance was only 240 MB, which downloads in only a few minutes for most network connections.

3. DISTRIBUTED VERSION CONTROL

Two important issues in using virtual appliances on students' computers are that instructional staff no longer have easy access to the students' virtual machines and students' computers are often not backed up. We need to provide a way for instructional staff to help students, test and grade homework assignments done in the virtual appliances, and limit the damage caused by students' computer failure.

We combine virtual appliances with a distributed version control system to provide reliable storage for students' homework assignments and manage the submission and grading of homework assignments. We use Git, the version control system used for many commercial open source software projects, including the Linux kernel. Git provides several advantages. First, it enables students to learn how to use a production version control system. Second, unlike systems such as CVS, it is designed from the ground up to support distributed version control for facilitating collaborative work among users that may be loosely connected in their work, which is particularly helpful for supporting distance learning students. There is no problem with offline work since each repository is self-contained and independent.

We provide a Git server for check ins and submissions. The system has modest computing and storage requirements, and can be used with any backed up machine as a server, including a virtual machine installed on an instructor's desktop computer. Students simply check in code to reliable storage, and do not need to back up their own computers for this purpose. The version control repository also enables multiple students to work together on group homework projects. We ask students to check in early and often. This helps the instructional staff to see who is doing the work in group programming assignments, and identify problems that students may be having well before the homework submission deadline so that we can help. We use the same system for submitting homework assignments, guaranteeing that students cannot tamper with submission dates and times. We also use the same system for returning graded homework assignments, whereby instructional staff can check in comments directly into students' code to provide better feedback to students. For our operating system assignments, students write C code. Our convention is to have students comment their code using `/* */` syntax to easily distinguish from instructional staff comments using `//` syntax.

We set up the Git server by providing each student a UNIX account on the machine using their university ID, but with only access to a specialized Git shell for running Git commands, namely `/usr/bin/git-shell`. Students connect to the server using SSH keys, which is secure and convenient. Each student repository is a clone from a template repository, where we provide the base file structure like the kernel

sources, or empty files to fill up. We optimize how repositories are stored since almost all programming assignments involve modifications to the Linux kernel tree, which can be large. By leveraging Git clone hardlink capability, the actual data is stored only once on the physical layer although each repository contains the kernel tree. Each repository stores only diffs from the template. Permission on repository is made with standard UNIX permissions. Each repository is `chmod 740`. Only the respective student and the instructional staff can access the repository. Repository corruption cannot happen since students have no access on the file system, and Git itself will not corrupt itself.

To guarantee that students cannot tamper with submission dates and times, we log all the pushes on the server, which contains the date of the push, the username, and the object hashes. This allows us to know who did what and when. In addition to logging, when students push their commits, they see in their console how much time is left until the homework submission deadline. Pushes after the deadline are rejected. We do not trust the timestamps in their commits, which can be manipulated since they simply reflect the timestamps of the students' computers.

4. LIVE DEMONSTRATIONS

Since instructional staff did not have easy access to students' virtual appliances, we could not test students' work directly in their virtual appliances. Having students check in their appliances instead of source code would incur too much storage overhead. We could build their programs and install, reboot, and test them in our own virtual appliances, but that would be time consuming for a large class.

We instead decided to require students to do live demonstrations of their kernel programming homework assignments. Students were assigned different 20 minute time slots during office hours to do their demonstrations. We automated the build process for the submitted code to generate a bootable kernel image for each homework submission. Students were given a clean virtual appliance and asked to install, reboot, and demonstrate the functionality of their kernel images. By starting with the submitted code, we ensured that the kernel images generated were based on the actual homework submissions. By having students demonstrate their submissions, we made grading those submissions easier since instructional staff did not have to manually perform the time consuming install, reboot, and testing for each submission.

More importantly, the live demonstrations provided a good opportunity for students to interact with the instructional staff and gain a better understanding of what they did right and what they did wrong in their programming assignments. Instructional staff could identify which students knew the material and which did not, and also which students really did the work and understood the material.

5. EXPERIENCES

We have taught operating systems at Columbia University for almost ten years using various forms of virtualization to support students working in groups of two or three to gain hands-on experience with Linux kernel development. We have used three methods for teaching and discuss and compare our experiences with each of them.

We started out using a shared computer laboratory with Linux desktop machines running VMware Workstation to support virtual machines. There was no shared storage infrastructure available to store the virtual machines, so they were stored locally on each desktop and statically assigned to students. Each time the course was taught, a computer support staff had to install the right version of VMware, create and distribute the virtual machines to each desktop, and deal with hardware failures and other issues as the desktop machines became out of date. While students learned a tremendous amount about operating systems, it was at times hard for students to work with the laboratory infrastructure. Even though we provided students with backup virtual machines allocated on other desktops, a desktop machine failure could cause a group of students to lose all their work on the virtual machines assigned to that desktop. Students lacked administrative privileges on the machines, and so could not fix the problems even if they knew how, instead having to wait until normal business hours before a support staff addressed the issue. This problem also occurred at times when students corrupted their virtual machine and lacked the access rights to install a clean virtual machine so they could start over. Finding funds to replace old or dead machines was often an issue. We did not use a version control mechanism at the time and relied on university submission tools, which were painful to use for both students and instructional staff. In particular, there was no way for students to verify the completeness of their submissions.

We have also used a server cluster consisting of a set of powerful multi-core servers and a dedicated SAN infrastructure for running VMware ESX server to support virtual machines. This infrastructure made it much easier to manage virtual machines, assign them to students, and dynamically allocate resources among virtual machines, but had some other problems. The more centralized infrastructure was easier to manage, but a configuration mistake or a hardware malfunction could bring down all of the virtual machines. The shared infrastructure became noticeably slow right before homework assignments were due when it was being used by everyone intensively at the same time. VMware snapshot functionality was very useful for reverting a corrupted virtual machine to a good state, but some students took so many snapshots that they filled up the entire multi-TB SAN. Computer support staff limited network access to the infrastructure for security reasons, making it more difficult for students to use their virtual machines.

Most recently, we have been using our virtual appliance and distributed version control solution to teach operating systems for the last year. The experiences of both students and instructors with this approach to teaching operating systems have been very positive. Students like running virtual appliances on their own computers, as they have full control over how resources are used, eliminating previous complaints about the hardware used to support the virtual machine infrastructure. Students also like no longer having to pay fees associated with using university computer facilities. The Git version control system provides experience with real-world software development tools and makes homework assignments much easier to manage. Students can check out what they submitted to make sure the submission works properly. Instructional staff can view check in logs and see how students are progressing, which students are contributing to group homework assignments and which ones are not, and can add comments to students' code bases directly to provide helpful feedback as part of the homework

grading process. Students like the increased interactions with instructional staff provided by having live demonstrations and code review of kernel programming assignments. Instructional staff find it easier to grade the assignments by relying on students to demonstrate their work instead of having to install, reboot, and test students' kernel programming submissions themselves. Our experience has been that the combination of virtual appliances, distributed version control, and live demonstrations has been a powerful pedagogical tool for teaching operating systems.

6. RELATED WORK

Many approaches have been explored for providing programming experience in operating system courses. Courses with user-level projects require only developing code designed to run in unprivileged mode. Examples of such projects include writing modules for a user-level simulator such as Nachos [2], user-level threads programming, or systems programming with a production operating system such as Linux, Solaris or Windows XP. While user-level projects do provide students with some hands-on experience with operating systems, they do not provide direct kernel-level development experience. As a result, user-level projects do not effectively address important issues such as bootstrapping, handling interrupts, the kernel-level development and debugging process, or understanding the kernel internals of a full-featured operating system.

Courses with kernel-level projects require writing or modifying code designed to run in supervisor mode. Kernel-level projects can provide a better pedagogical vehicle for learning about real-world operating system design and implementation. Some are based on a small pedagogical operating system like MINIX [8] or Pintos [6], or involve a pedagogical operating system to be used only in a simulated environment [3]. Pedagogical operating systems are smaller than production operating systems and do not expose students to many of the real-world issues that arise in practice. Because pedagogical operating systems are not used in practice, keeping them from becoming dated can require substantial effort and they may have more limited lifetimes due to changes in technology and operating system practice [3].

Other courses involve kernel-level projects with production operating systems. One of the authors developed the first course to leverage virtual machines for teaching operating systems with Linux kernel projects [4], an approach which has since been widely adopted. This work was initially done using a shared computer laboratory [5]. Others have built on this work by also using a server cluster [1]. We build on our prior work to provide a solution that does not require providing or maintaining significant computing infrastructure, and introduces course management techniques based on distributed version control and live demonstrations to improve the pedagogical value of the course.

Production version control systems such as CVS have been used in operating system courses [3] for supporting source code development, but not for managing homework submissions and facilitating grading. CVS has also been used for homework submissions in lieu of homebrew university submission tools [7], but required use of university computer laboratory facilities, did not provide effective safeguards against repository corruption, and did not support the requirements of operating system courses with kernel-level programming projects. Unlike these approaches, we use Git to provide distributed version control management and reliable storage without using and maintaining university computer laboratory facilities. Git provides a better, more scalable model than CVS for reliable and consistency check ins for kernel development, resulting in its adoption by the Linux kernel developer community. We also show how Git and live demonstrations can be used to provide better grading mechanisms for homework assignments and improved feedback to students to enhance their learning.

7. CONCLUSIONS

We have developed a new approach for teaching operating systems that provides hands-on kernel-level project experience without the need for computer laboratory facilities for teaching purposes. Our solution creates a virtual appliance for doing operating system homework assignments which can be easily deployed and run on students' personal computers in virtual machines without interfering with any existing software already on the students' computers. We combine virtual appliances with a distributed version control system to provide reliable storage for students' homework assignments, support students working together on group homework assignments, and manage the submission and grading of homework assignments. Live demonstrations are also used to simplify grading and provide better feedback to students. We have used this approach to teach operating systems to both undergraduate and graduate students, and both on-campus and remote distance learning students. Our experiences demonstrate that this solution is cost-effective, easy to deploy and use, and improves the educational experience of students in learning about operating systems.

8. ACKNOWLEDGMENTS

This work was supported in part by VMware and NSF grants CNS-0426623, CNS-0717544, and CNS-0914845.

9. REFERENCES

[1] J. C. Adams and W. D. Laverell. Configuring a Multicourse Lab for System-Level Projects. In *Proceedings of SIGCSE 2005*, St. Louis, MO, Feb. 2005.

[2] W. Christopher, S. Proctor, and T. Anderson. The Nachos Instructional Operating System. `http://cs.berkeley.edu/~tea/nachos/nachos.ps`.

[3] D. A. Holland, A. T. Lim, and M. I. Seltzer. A New Instructional Operating System. In *Proceedings of SIGCSE 2002*, pages 111–115, Feb. 2002.

[4] J. Nieh and Özgür Can Leonard. Examining VMware. *Dr. Dobb's Journal*, pages 70–76, Aug. 2000.

[5] J. Nieh and C. Vaill. Experiences Teaching Operating Systems Using Virtual Platforms and Linux. In *Proceedings of SIGCSE 2005*, St. Louis, MO, Feb. 2005.

[6] B. Pfaff, A. Romano, and G. Back. The Pintos Instructional Operating System Kernel. In *Proceedings of SIGCSE 2009*, Chattanooga, TN, Mar. 2009.

[7] K. L. Reid and G. V. Wilson. Learning by Doing: Introducing Version Control as a Way to Manage Student Assignments. In *Proceedings of SIGCSE 2005*, St. Louis, MO, Feb. 2005.

[8] A. Tanenbaum. A UNIX Clone with Source Code for Operating Systems Courses. *Operating Systems Review*, 21(1):20–29, Jan. 1987.