Configuring a Multi-Course Lab for System-Level Projects

Joel C. Adams W. David Laverell
Department of Computer Science
Calvin College
Grand Rapids, MI 49546
1-616-526-8562
{adams, lave}@calvin.edu

ABSTRACT
Having students modify an actual operating system kernel or network protocol stack opens their eyes to what is going on "beneath the hood" of a computer. However student modifications to a system may result in an unstable computer. Because of this, giving students such experience has in the past required a lab and/or computers dedicated to the students in the system-level course, and computer science departments without such dedicated facilities have been unable to provide their students with system-level experience. In this paper, we present two ways of giving students system-level experience in a non-dedicated lab; one using commercial software (VMWare), and another using open-source freeware (User Mode Linux Kernel).

Categories and Subject Descriptors
K.3 [**Computers & Education**]: Computer & Information Science Education – *Computer Science Education*.

General Terms
Management, Design, Reliability, Experimentation,

Keywords
Laboratories, Operating Systems, Networking, User Mode Linux Kernel, VMWare.

1. INTRODUCTION
Most laboratory and programming projects are intended to provide students with a concrete, hands-on experience to help them better understand abstract concept(s). In computer science courses where students are studying system-level concepts, it may be desirable to give students system-level experiences, such as requiring them to modify a real kernel in an *Operating Systems* course [8], the actual protocol-stack in a *Computer Networking* course [9], and so on. Because students must have access to the system's source code, projects using a real kernel or stack are limited to laboratories that use an open-source system, such as Linux, FreeBSD, OpenBSD, etc.

SIGCSE'05, February 23–27, 2005, St. Louis, Missouri, USA.

A student-modified system tends to be unstable until the student successfully completes their project. Since it is inconsiderate to subject students from other courses to such instability, system-level projects have in the past required a dedicated computing laboratory, separate from other general-purpose laboratories [2][5]. Because of this, computer science departments with a single laboratory have often used hardware emulators [3] that provide less authentic experiences.

Even at institutions with separate, dedicated laboratories for systems courses, the following issues must be addressed:

1. *The instability of a student-modified system makes that system unusable by anyone else*. Two mechanisms for resolving this problem include: (i) assign each student their own machine and system to modify; or (ii) have multiple students share a specific machine, whose disk contains a separate, bootable partition for each student assigned to it. Both of these solutions require a student to always use the same machine.
2. *If the number of students N exceeds the number of available machines M, then an instructor must allocate the M machines among the N students*. Common ways to resolve this issue include: (i) having multiple students use the same machine, or (ii) having students work in groups of size N/M. (If N » M, then multiple *groups* may have to use the same machine.)
3. *If a laboratory configuration requires each student (or group) to use a specific machine for their project, then students (or groups) may have to contend with one another for that machine*. A common mechanism for resolving this problem is a *reservation system*, in which students (or groups) are required to sign up for a particular time-slot during which they will have sole access to their machine.
4. *If a laboratory configuration requires each student (or group) to use a specific machine for their project, then a student (or group) will be unable to work on their project while their machine is in use by another student (or group), even if other machines are open*. The only obvious means of resolving this problem is to not require each student (or group) to use a particular machine, but doing so creates a conflict with the solutions for issue 1 above.

In a nutshell, the problem is this: to limit the potential instability of student-modified systems (or the abuse of super-user privileges), students doing system-level projects have in the past been required to work on a particular machine. However most departments have insufficient resources to provide each student with their own machine, making machine-sharing a necessity. As a result, students lose flexibility as to *when* they may work. This is especially inefficient if other machines are sitting idle.

In this paper, we present a solution to all of these problems. Our solution can be used in either a general-purpose lab or a dedicated lab, making it applicable at virtually any institution. As such, it makes system-level projects possible at institutions where they were impractical in the past. It also makes such projects easier to manage at other institutions.

2. BACKGROUND

The recent development of software like *VMWare* [10] creates new possibilities for resolving the four issues listed in Section 1. VMWare provides an x86-emulating *virtual machine* that allows a user to run another operating system "on top of" the machine's real operating system, as shown in Figure 1:

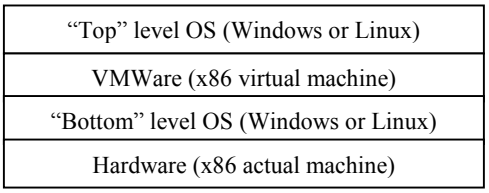

Figure 1. VMWare

VMWare works by allowing a user to set up one or more *virtual disks*. Each virtual disk can contain its own operating system, that may be the same as or different from the "bottom" level operating system. To run VMWare, one simply specifies a virtual disk; the virtual machine then begins running and loads whatever operating system (or boot loader) it finds on that virtual disk. If for some reason the "top" level system should lock up (e.g., because of a student's modifications), the "bottom" level OS is safely isolated from the problem. Recovery from a disaster as thus as simple as restarting VMWare – the real "bottom" level OS need not be rebooted.

When it starts, VMWare loads the specified virtual disk into a RAM-disk which it treats as its boot disk. When VMWare is shut down, if its RAM-disk has been modified, it asks the user if they want to (i) save the changes, (ii) discard the changes, or (iii) defer the decision. Thanks to this mechanism, any changes one makes to the system remain local to the RAM-disk, until the user saves them to the virtual disk. This allows a user to thoroughly test any modifications they've made to their system before they actually alter their virtual disk.

In [7], Jason Nieh describes a dedicated operating systems laboratory in which each machine was equipped with VMWare. ([6] describes a similar networking laboratory.) While Nieh's laboratory resolved issues 1, 2, and 3, it failed to address issue 4; student-groups had to use particular machines, and if their machine was in use by another group, they were forced to wait even if other machines were vacant.

This paper describes our *Systems Lab*, a laboratory that also uses VMWare to resolve issue 1. However our laboratory uses an alternative approach to resolve issues 2 and 3, that also resolves issue 4. In addition, our approach does not require a dedicated laboratory; it can be used in either a dedicated- or a shared-lab environment.

3. VMWARE + FAST ETHERNET + NFS

To resolve issues 1-4, we designed a shared *Systems Lab* in which we could assign system-level projects in our *Operating Systems* and *Computer Networking* courses. For lab machines, we purchased 25 off-lease 3-year-old PCs. Each PC had a 450-MHz Pentium-II CPU, 128 MB RAM (which we upgraded to 256MB), a 9 GB hard disk, and a 10-100Mbps Ethernet card.

For a network, we connected machines into "workgroups" of 3 via a 5-port 100 Mbps Ethernet switch, each of which was then connected to a central 24-port switch (for future expansion). By configuring a 25th PC as both a firewall and an HTTP proxy server, and using it as the connection between our switch and our campus network, we were able to isolate our lab from the outside world, but still provide each PC with web-access.

Our *Operating Systems* and *Computer Networking* courses both meet during the same semester, and the enrollment in each is greater than the number of machines in our lab. To deal with the resulting congestion, we felt it was imperative to resolve all four of the issues described in Section 1, so that each student or group could work at any open machine.

To resolve issue 1, we installed VMWare on each machine, and used it to run Linux on top of Linux, as shown in Figure 2:

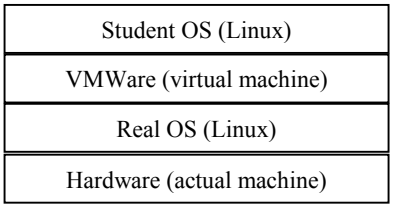

Figure 2. Linux on top of Linux via VMWare

For the "top" level OS, we prepared a 2 GB VMWare virtual disk containing a full Linux installation. This virtual disk was saved in a public directory, with its permissions set to read-only.

To resolve issues 2, 3, and 4, we added a 26th PC as a file server, plus a RAID array containing four 120GB disks. Configured as a RAID-5 device, this array provides 360GB of usable disk space – enough to give 180 students 2 GB of disk space each.

To provide students with home directories, we created a Linux `/home` partition on this RAID array, and created student home directories within `/home`. Each lab PC was configured to mount `/home` via the Network File System (NFS) service, and student accounts were managed via the Network Information Service (NIS) from the file server. The resulting configuration is shown in Figure 3, below.

With home directories in place, students were instructed to copy the read-only virtual disk from its public directory to their home directories, and were given the root password to the OS on the virtual disk. (This password was different from that of the "bottom" level OS). Since every PC in the lab accessed their account information via NIS and mounted `/home` via NFS, students could now login to any PC in the lab, and then boot VMWare using their own virtual disk. From any PC in the lab, they could modify their kernel, recompile its source code, save changes, and so on, without affecting anyone else's virtual disk or the "bottom" level OS.

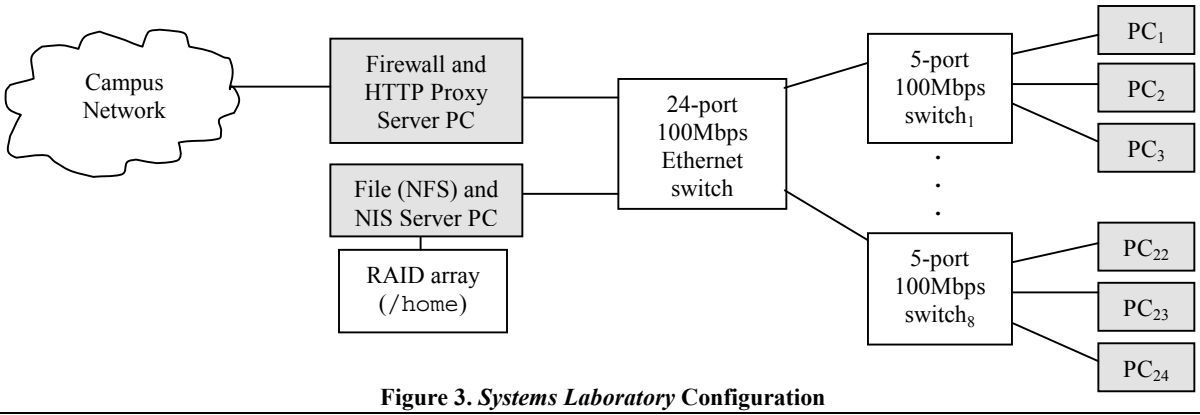

Figure 3. *Systems Laboratory* Configuration

It required some time and effort to get all of this working properly. However, this approach has let us realize these benefits:

1. It resolves issue 1 by allowing students to work on system-level projects. Students using the lab may be in the same or in different courses. Different virtual disks can be created for different courses, as necessary or appropriate.
2. It resolves issues 2, 3, and 4 in the same way as any other lab – students may work at any open machine.
3. It lets instructors assign group or individual projects as pedagogically appropriate, rather than as a work-around necessitated by limited computing resources.
4. It lets students test the changes they make to their system before they save those changes to their virtual disks (or defer making the decision).
5. If students manage to irreparably mangle their virtual disks, it lets them easily start over by recopying the original read-only virtual disk from the public directory into their home directory.
6. Aside from removing the student accounts, no special action is required to restage the machines, either at the end of the semester, or the next time the course is offered, saving time in the long run.
7. Although we used this approach in a separate Systems Lab, there is nothing that prevents this approach from being used in a general-access lab. This solution can thus be used at departments or institutions that have a single computing laboratory.

Because of these benefits, we believe the time spent to get this arrangement working was time well spent.

4. OBSERVATIONS

After having used the Systems Lab in our curriculum, we have the following observations:

- Students enjoyed being able to "get down and dirty" with the low-level system source code. Many expressed amazement at the ugly *spaghetti code* (i.e., extensive use of C's `goto`) they encountered in the Linux kernel.
- Students appreciated VMWare's permitting them to test out their system modifications before committing them to their virtual disks. Just two *Operating Systems* students (out of roughly 30) mangled their hard disks by neglecting to use this feature. All these students had to do to was recopy the original read-only virtual disk from the public directory into their home directory, and they could work again.
- Students rose to the challenge of doing system-level projects that no previous class had been assigned. Having to wrestle through issues with which older students could not help created a certain *esprit de corps* in the course.
- Our 100 Mbps (fast) Ethernet was fast enough to provide adequate performance. VMWare could take a few seconds to start up, as it accessed a virtual disk across the network via NFS. However once it was running, very little speed difference was discernable between it and the local machine.
- If many students started VMWare simultaneously, our PC file server could bog down under the load of downloading so many virtual disks at the same time. To remedy this, we have since replaced our PC file server with a dual 2.4GHz Xeon multiprocessor, which has solved the problem.
- Compiling the 2.2 Linux kernel required about 10 minutes on our 450MHz Pentium-II CPUs. This meant students could no longer use the compiler as a substitute for careful design, coding, and debugging, as some were in the habit of doing. For most students, this was a new experience and a revelation. It was also the biggest source of student complaints. To compile the 2.4 kernel in a similar length of time, we have recently upgraded to 1GHz Pentium-IIIs.
- Students from different system-level courses (*Operating Systems* vs. *Computer Networking*) used the lab simultaneously without interfering with one another. In the second semester, students in two different system-level courses (*Computer Security* vs. *Network Administration*) used the lab without interfering with one another.
- Though the Systems Lab is a dedicated lab, this approach could be used in *any* lab shared by multiple courses. We are presently considering adding it to our general-access labs to provide greater flexibility in where students work.

5. USE IN OTHER COURSES

Our *Operating Systems* course used the Systems Lab for modified (updated) versions of some of the assignments found in [8]. Other courses and activities using the lab include:

- *Computer Networking*, which used the lab for exercises in TCP/IP socket programming, packet sniffing, protocol study, and so on. This course met concurrently with our *Operating Systems* course during the fall semester.
- *Computer Security*, which used the lab for exercises in password breaking, attack methodologies, intrusion detection, and so on. VMWare also allows the creation of a *virtual honeypot* [1] – a honeypot installed on a "top" level OS – that completely hides the "bottom" level OS from an intruder, allowing the "bottom" level OS to surreptitiously monitor the intruder's behavior without being detected.
- *Network Administration*, which used the lab for network administration exercises including configuration of network services, routers, firewalls; network analysis; and so on. This course met concurrently with our *Computer Security* course during the spring semester.

Each course is an upper level elective in our curriculum.

6. BUDGET

Table 1 (below) presents the cost of building our lab:

Table 1. Systems Lab Expenses

Item	Qty.	Unit Cost	Total Cost
3-Year-Old Used PCs	26	$150	$3,900
128MB Memory Upgrades	26	$25	$ 650
VMWare Licenses	25	$113	$2,825
VMWare Support	25	$24	$ 600
5-port Fast Ethernet Switch	8	$40	$ 320
24-port Fast Ethernet Switch	1	$800	$ 800
RAID Array (Promise RM8000)	1	$3415	$3,415
Total			**$12,510**

Our original 3-year-old PCs were 450 MHz Pentium-IIs. As mentioned above, we have since replaced them with (3-year-old) 1 GHz Pentium-III machines, at the cost shown in Table 1.

We could have either purchased a smaller central switch (e.g., 16-port), or purchased a larger central switch (e.g., 32-port) and eliminated the eight 5-port switches. We chose the configuration above for its flexibility and expandability.

Some of these prices have already decreased. However these prices should provide an approximate cost for building a laboratory with capabilities similar to (or better than) ours.

In addition to the items in Table 1, we purchased additional networking hardware specifically for our *Computer Networks, Computer Security,* and *Network Administration* courses, including a hub and router for each "work group", cable tray, patch cables, and so on. Since these are not directly required to give system-level projects, we have omitted their cost here.

7. USER MODE LINUX KERNEL

At the risk of stating the obvious, VMWare is not inexpensive – its licenses and support account for more than 27% of the expenses in Table 1, and push its total cost above $10,000. For many institutions, this extra expense may be prohibitive and prevent a budget officer from approving funds for a lab.

Thankfully, there is a freeware alternative to VMWare called **User Mode Linux Kernel (UMLK)** [4]. UMLK allows a user to run a Linux kernel at the user level. As such, it provides a way to give students a system-level programming experience, and since it is free, it can significantly reduce the cost of doing so (compared to VMWare). We hope to someday read a SIGCSE paper reporting on the use of UMLK in assigning system-level projects.

Where VMWare is an x86 hardware emulator (i.e., a virtual machine) that boots from a virtual disk, UMLK is a kernel with its own file system that runs directly on top of another Linux kernel, as shown in Figure 4:

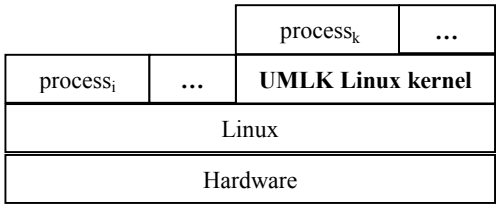

Figure 4. UMLK Structure

Because of these differences, UMLK is not quite as flexible or full-featured as VMWare. At present:

- UMLK runs only Linux on top of Linux – neither the "top" nor the "bottom" level operating system can currently be any other OS. This may limit its applicability at some institutions.
- Where VMWare allows you to install any x86-based OS on your virtual disk, UMLK runs whatever Linux kernel is installed in its file system. Different kernels can be run at different times in UMLK, but changing the kernel requires that the new kernel be (re)installed into the file system. By contrast, switching kernels in VMWare is simply a matter of booting from a different virtual disk on which the new kernel has been installed – one can easily revert to a previous kernel without having to reinstall it.
- Where VMWare allows a user to test system modifications before committing them to their virtual disk, UMLK users must commit such modifications to their UMLK file system before testing. One might argue that if students have never been "spoiled" by using VMWare, they will never know what they are missing. However the absence of this mechanism significantly increases the cost of helping a student recover from a catastrophic mistake, since a new kernel must be reinstalled on their file system to recover from such an error.

UMLK thus provides a viable alternative approach for giving system-level projects in a (shared or dedicated) lab setting. It may be attractive at institutions with existing laboratories running Linux, or who are willing to add a dual-boot option in a laboratory running Windows. Compared to VMWare, it trades off some convenience and flexibility for expense.

8. CONCLUSIONS

The commercial product VMWare provides a robust, flexible, and convenient means of building an environment in which students can work on system-level projects. By isolating the system the student is working on from the operating system running on the actual hardware, VMWare permits a student to safely modify their "top" level system, without danger of the "bottom" level system being compromised and rendered unstable. VMWare also allows students to test changes they've made to their systems before permanently committing those changes. VMWare thus provides a convenient tool by which students can work on authentic system-level projects, either in a dedicated systems lab, or in a general-purpose computer lab.

For institutions seeking a less expensive solution, the non-commercial product User Mode Linux Kernel provides a viable alternate means of providing students with an authentic system-level programming experience, though at the price of some features and flexibility available in VMWare.

For institutions where the general-purpose lab is already heavily utilized, we have demonstrated how to build a separate, dedicated Systems Lab for use in operating systems, networking, security, and other "low level" courses. By having students access their VMWare virtual disks via NFS over a fast network from a centralized file server, a student can work on a system-level project from any open machine, and students in different "low level" courses can share the same laboratory with a minimal level of conflict.

We hope that our experience will provide a useful model for other institutions seeking to build their own facilities.

9. REFERENCES

[1] R. Barnet, Monitoring VMWare Honeypots, http://honeypots.sourceforge.net/monitoring_vmware_honeypots.html

[2] M. Claypool, D. Finkel, C. Wills, An Open Source Laboratory for Operating Systems Projects, *Proceedings of 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE),* June 2001, pp. 145-148.

[3] J. Dickinson, Operating Systems Projects Built on a Simple Hardware Simulator, *Proceedings of the 31st SIGCSE Technical Symposium*, March 2000, pp. 320-324.

[4] J. Dike, The User Mode Linux Kernel Home Page, http://user-mode-linux.sourceforge.net/.

[5] J. Hill, C. Carver, J. Humphries, and U. Pooch, Using an Isolated Network Laboratory to Teach Advanced Networks and Security, *Proceedings of the 32nd SIGCSE Technical Symposium*, February 2001, pp. 36-40.

[6] B. Kneale, and I. Box, A Virtual Learning Environment for Real-World Networking, *Proceedings of the Informing Science + IT Education (InSITE) Conference*, June 2003, pp. 671-683.

[7] J. Nieh and O. Leonard, Examining VMWare, *Dr. Dobb's Journal*, August 2000, pp. 70-76.

[8] G. Nutt, *Kernel Projects for Linux*, Addison Wesley, 2001.

[9] B. Richards, Teaching Network Protocols Through Debugging, *Proceedings of the 31st SIGCSE Technical Symposium*, March 2000, pp. 256-260.

[10] VMWare, http://www.vmware.com/.