Examining VMware

Dr. Dobb's Journal August 2000

By Jason Nieh and Ozgur Can Leonard

Jason is an assistant professor of computer science at Columbia University. He can be reached at nieh@cs.columbia.edu. Can was a teaching assistant for the operating-systems course at Columbia. He can be reached at ocl3@cs.columbia.edu.

VMware is a virtual-machine platform that provides an abstraction of x86 PC hardware so that multiple operating systems can run unmodified and at the same time on a standard PC. For developers, this means you can run multiple development environments on your desktop without rebooting or repartitioning. In the process, you can isolate and protect operating environments (and the applications and data that are running in them), as well as interoperate among operating systems for networking, device/file sharing, and cut-and-paste. For users, VMware makes it possible to run Windows applications with Linux. VMware comes in two flavors, depending on the operating system running on the user's Pentium-based (or compatible) PC -- VMware for Linux, and VMware for Windows NT/2000. VMware installs like an application program, requiring no special hardware support.

In this article, we'll present a different perspective by discussing our experiences teaching operating-systems courses at Columbia University (http://www .cs.columbia.edu/~nieh/teaching/w4118). Using Linux and VMware, we were able to give students a flexible virtual kernel-level development environment in which operating systems can be developed, debugged, and rebooted in a shared computing lab environment without affecting other application users.

Projects in operating-system courses typically can be categorized as user level or kernel level:

- 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 (see The Nachos Instructional Operating System, http://http.cs.berkeley.edu/~tea/nachos/ nachos.ps), user-level threads programming, or systems programming with a commercial operating system such as Solaris or Windows NT.
- Kernel-level projects require writing or modifying code designed to run in supervisor mode. Examples of such projects include writing a small kernel from scratch, studying the internals of a pedagogical operating system such as Minix (http://www.cs.vu.nl/~ast/minix.html), or possibly modifying the kernel of a commercially available, full-featured operating system.

The latter is made possible due to the increasing popularity and maturity of freely available open-source operating systems such as Linux and BSD variants.

OS courses with only user-level student projects are more common because the projects are typically no more difficult to set up and administer than other user-level applications. While user-level projects provide students with some hands on experience, they don't provide direct kernel-level development experience. Consequently, 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.

Kernel-level projects, on the other hand, provide a better pedagogical vehicle for learning about real-world operating- system design and implementation. However, students must be given root privileges to do many of the things required for kernel development, such as installing new kernels, rebooting, kernel testing and debugging, and so on. In addition, students typically need exclusive access to a computer so that the development cycle of plan-implement-reboot-test-debug can be done without inconveniencing others. For a typically large introductory OS class, providing every student with a computer on which to run as root to do kernel development is difficult to administer and prohibitively expensive. As a result, most courses can only offer user-level project alternatives.

The VMware Virtual Machine

Again, VMware (http://www.vmware.com/) is a virtual-machine platform that makes it possible to run an unmodified operating system as a user-level application. The OS running within VMware can be rebooted, crashed, modified, and reinstalled without affecting the integrity of other applications running on the computer. In the operating-systems class at Columbia University, we used VMware to provide students with a virtual kernel-development environment in which to work on Linux kernel-level projects without requiring a separate computer for each student. VMware made it possible for us to teach operating systems with kernel-level projects to a class of 140 students, using nothing more than a shared PC lab that was simultaneously used by other students for general computing needs.

VMware is a virtual-machine monitor that grew from OS research at Stanford University. A virtual-machine monitor is an additional layer of software between the hardware and the operating system that virtualizes all of the hardware resources of the machine. It essentially creates a virtual hardware execution environment called a "virtual machine" (VM). Multiple VMs can be used at the same time, and each VM provides isolation from the real hardware and other activities of the underlying system; see Figure 1. Because it provides the illusion of standard PC hardware within a VM, VMware can be used to run multiple unmodified PC operating systems simultaneously on the same machine by running each operating system in its own VM. An OS running as a user-level application on top of VMware is called a "guest OS." The native OS originally running on the real hardware is called the "host OS."

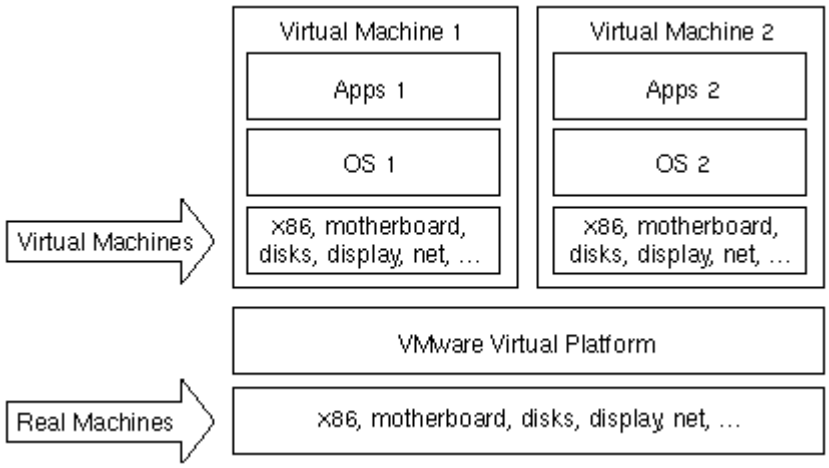

Figure 1: The VMware virtual platform.

VMware is low-level enough to make a guest OS appear to be receiving hardware interrupts (such as timer interrupts) and behave as if it were the only OS on the machine. At the same time, it provides isolation so that a failure in or misbehaving of a guest OS does not affect other guest OSs or the underlying system. For instance, a guest OS crashing will not crash the underlying system. As opposed to a software simulator, much of the code running in a VM executes directly on the hardware without interpretation. Only privileged instructions are trapped and impose additional overhead. The major advantage of using a VM as opposed to a simulator is the performance improvement possible through direct execution of unprivileged instructions.

Operating systems currently supported as guest operating systems under VMware include Windows 95/98/2000/NT, FreeBSD, Solaris, Novell Netware, DOS, and Linux, all of which run unmodified. Theoretically, any OS that can run on an x86 architecture can run as a guest OS, since it will see a complete virtualized PC environment. For host operating systems, VMware currently runs and is supported on Windows 2000/NT and Linux. In addition, there is an unsupported port that can use FreeBSD 4.0 as the host OS. We used the Linux version of VMware for the operating-systems course we taught.

The virtual-machine monitor in Linux is implemented via a loadable kernel module that provides the necessary hooks for hardware emulation, interrupt generation, and other low-level details. There is also a GUI, which is the visual interface to the virtual machine, and makes VMware look like a real computer from the moment it boots the VM.

Teaching Operating Systems Using VMware

For the last few years, the introductory operating-systems course at Columbia University has been taught using the user-level Nachos simulator. As is typical at many universities, the computer lab facility we had available for the course could not be used as a dedicated operating systems lab. Nachos was easy to set up and administer in a computer lab shared among multiple courses.

When VMware was introduced, we thought it would be a great tool for teaching operating systems because it would let us provide kernel-level student projects in the shared computer lab available for the course. To give us the opportunity to try VMware for our instructional purposes, VMware provided us with evaluation licenses for the course. We used VMware 1.03, which was the latest product version available before the course started. More than 140 students enrolled in the course, a 50 percent increase over previous years.

VMware allowed us to configure a guest OS as a kernel development environment for a group of students. We were able to give students root access to guest OSs running in VMs without compromising the security of the lab machines. Because faults that occur in running the guest OS are contained within its respective VM, students could crash and reboot their guest OSs without interfering with the operation of the host OS. It was thus possible to retain the shared status of the computer lab we used, because students working on other projects could be sharing the same computer as a group debugging their kernel.

Using VMware, our operating-systems class focused on the kernel internals of Linux. Students were able to learn in-depth about the kernel design and implementation of a widely deployed operating system, and gained valuable software experience in working with a large body of source code. The experience gained by such work is directly applicable to the kind of operating-systems work that students will encounter in the industry. We chose Linux in particular due to the extensive online documentation available, as well as the immense popularity it enjoys. Students were more interested in doing the projects because they were working with something they felt was practical and relevant. The kernel-level projects we assigned were:

- Familiarization with the kernel and system calls. Students learned how to build and run an operating system. They then modified the behavior of an existing system call and created a new system call.
- Schedulers. Students implemented two new weighted round-robin schedulers, and evaluated their effectiveness as alternatives to the regular Linux scheduler.
- Memory management. Students modified the Linux memory management internals to change the criteria used to determine how much main memory a process could use and what pages were swapped out.
- File systems. Students modified the block allocation scheme used in the ext2fs file system, and observed the resulting performance differences.
- Device drivers. Students implemented a kernel device driver, dev/clock, as a loadable kernel module interfacing to the gettimeofday/settimeofday system calls.

VMware was indispensible for the implementation and testing of the project assignments, and provided a framework in which students were able to observe the correctness and performance of their implementations.

VMware Setup

The shared computer lab we used for the course included 24 student machines: 12 333-MHz Intel Pentium II PCs with 128 MB of RAM, and 12 266-MHz AMD K6 PCs with 64 MB of RAM. Each machine had a 4 GB local disk and ran Red Hat Linux 6.0 with the X Windowing system. All of the machines NFS-mounted users' home directories from a separate set of file servers. Given the class size and the number of machines available, we conservatively grouped students into 45 project groups of 3-4 students, with two groups assigned to each machine. Each group was given its own VM and guest OS.

The process of setting up VMware and the guest OS on a single Linux machine is:

1. Download the VMware for Linux binaries and kernel modules.

2. Install VMware, which needs to be done as root in order to load the kernel modules.

3. Obtain a VMware license and save the license file in your home directory.

4. Start VMware and follow the step-by-step instructions provided by the VMware GUI to define a VM hardware configuration to export to the guest OS.

5. Insert the installation media (floppy, CD-ROM) for the guest OS and press the Power On button in the VMware GUI.

The installation for the guest OS then proceeds in the same way as it would for installing the OS directly on the computer. The VMware VM configuration is saved to a plain-text config file that may later be modified without having to reinstall the guest OS. We discuss later the various VM configuration options for disk, memory, and networking, and explain the options we used to ensure proper security in a shared lab environment.

The VM disk configuration options determine how disk partitions mounted by the host OS are used to export a virtual storage device to the guest OS. VMware lets the guest OS mount raw disk partitions or use virtual disks. A virtual disk in VMware is simply a large file in the host OS file system that is treated by the VM as an IDE disk. We configured the VMs to use virtual disks so that disk problems caused by a misbehaving guest OS that a student installed would not affect the host disk partitions. The use of virtual disks made it simpler for students to reinstall a guest OS in the event of a disk crash in the VM. Because a virtual disk is just a regular file on the host OS file system, we just provided clean versions of the virtual disk so that students could use them to overwrite their own in case of unrecoverable disk crashes in the VM. VMware also has an option of marking virtual disks and raw partitions as "undoable," which lets changes be discarded at the end of a VMware session. This can prevent unwanted and accidental modifications to the guest OS file system from becoming permanent.

We allocated two 500 MB local disk partitions on each machine, then created a virtual disk in each partition. The 500 MB of disk space was necessary for providing a Red Hat Linux kernel development environment. The Linux development tools and source code required roughly 400 MB of space. This did not include the X Window system, which

was not necessary in the guest OS installation for our class projects. We installed the virtual disks on the local disks as opposed to the NFS-mounted file servers for performance reasons. In addition, the local disks were not backed up due to the network, and disk-space resources that would be required to do this for each of the 45 student project groups. Restricted access to the virtual disks was achieved using user groups. The lab system administrator created a number of user groups for the course, and set permissions on the virtual disks such that they could only be accessed by the teaching staff or members of the respective user group. Students were each assigned to one user group. As a result, even though it was possible to start VMware from any of the machines in the lab, virtual disks were protected from unauthorized access via standard Linux file permissions.

The VM memory configuration options determine the amount of host system memory that the VM is allowed to use. This option is one of the primary determining factors in the performance of the guest OS environment in the respective VM. To simplify administration, we used the same VM memory configuration for all of the Linux machines, despite the fact that some of the machines had more memory than others. Since some of the Linux machines had only 64 MB of RAM and needed to support two VMs as well as other users, we configured each VM with only 16 MB of RAM. More recently, all of the Linux machines have been upgraded to 128 MB of RAM. If the extra memory had been available when the class was taught, we would have configured each VM with 32 MB of RAM for better performance.

The VM network configuration options determine the type of networking available to the guest OS. The options are no networking, host-only networking, and bridged networking. The no-networking option does not export a network interface to the VM. The host-only networking option exports a network interface to the VM that only allows communication between the VM and the host machine. Under bridged networking, the host OS acts as a bridge between the VM and the LAN, effectively allowing the VM to run as a real networked machine with an IP address. To let students access their files on the host file system and the NFS-mounted file servers, we configured the VMs for host-only networking. Using host-only networking, students were able to use FTP to backup their work onto their regular home directories, which alleviated many of the problems of a virtual disk crash while working on an assignment. We did not provide full bridged networking to the VMs because of the security implications of allowing students to have root access on a full networked machine on a LAN.

We found the configuration of the guest Linux OS we used to be fairly straightforward. Since VMware emulates standard devices such as IDE disks and PCI Ethernet cards, installing Linux as a guest OS was easier than installing Linux on a typical PC, which may have nonstandard hardware devices. Because the VMware virtual devices do not necessarily correspond to physical devices (in terms of brand, for instance), it is possible to run Linux on VMware on a machine with nonstandard hardware that would have otherwise been unsupported by Linux.

To simplify the administration of multiple machines, the VM configuration and guest OS installation were done once on a single machine to create a master system. The master was then distributed to the other machines using rdist file- distribution software.

Experiences

Students' experiences with VMware were quite positive. Once VMware and the guest OS were installed, a user could simply turn on the guest machine by clicking on the Power On button in the VMware GUI. When running, the guest machine looks like a complete desktop computing environment running within a VMware window on the host OS. In addition to this window mode of operation, full-screen mode for the guest OS is also available. Shutdown and reboot of the guest machine also had the same look-and-feel of the host OS. A Power Off button turns the VMware system off in the same way that it does on normal hardware. Students found VMware easy and intuitive to use.

One feature students found quite useful was the ability to launch and use VMware from a remote machine. When used in window mode, VMware runs essentially like a normal X application under Linux, so it can be run such that the VMware GUI is displayed on another machine. Consequently, students did not have to compete for console access to use VMware. We also gave students who wanted remote-access functionality the option of using the X version of VNC, a thin-client software system developed by AT&T research. VNC (http://www.uk.research .att.com/vnc/download.html) provides screen-sharing technology so that users can see and control the exact same screen on multiple machines, which made it much easier for students to collaborate in their project groups. In addition, we found that VNC could be used by a number of distance learning and off-campus students in the industry who could not access VMware using X because of corporate firewalls.

In some cases, there were problems with using VMware together with VNC because of limitations in VNC's X protocol support. However, the combination of the two was robust enough that a distance-learning student in Japan was able to successfully collaborate with local students on their group assignments using the VMware/VNC system. Some remote users did have an initial problem using VMware remotely in which keystrokes would repeat randomly. Linux defines a keyboard delay time that determines the amount of time that a key must be pressed before it will start to repeat. The keyboard delay time can be changed using the Linux kbdrate command, and this can be done with the OS boot up by including the kbdrate command in the rc.local file. In most cases, increasing the keyboard delay time of the guest Linux OS from the Linux default value to its maximum value completely eliminated the unwanted repeating keystrokes.

A limitation of X and VNC is that they do not perform well over low-bandwidth connections such as modems. Students who only had low-bandwidth network access found the ability to Telnet to the running guest OS from the host OS to be useful. These students were still able to do project work through a Telnet window. However, VMware and the guest OS have to already be running for the students to Telnet to the guest OS. It is not currently possible to run VMware and start up the guest OS without running the

VMware GUI and clicking the Power On button. As a result, Telnet users had to rely on other users with fast enough network connections to power up and shut down the VM. A VMware feature to allow users to start up a VM and guest OS configuration without using the GUI would have been useful for the class. We discussed this with VMware and the company will provide this option in future releases.

A limitation of the version of VMware that we used was the lack of shrinkable virtual disks. Removing files from within a virtual disk in a guest OS did not cause the file image on the host OS to shrink. Students ran into problems with this when using an undoable virtual disk and committing changes at the end of a session to the virtual disk. The undoable disk option creates log files of disk changes that are not saved to disk until they are committed by users. If the log files were large enough, a commit operation could cause the host OS disk partition to run out of space, which would in turn leave the guest OS disk in an inconsistent state. Since the host OS disk partitions only had about 100 MB of extra space after the guest OS was installed, some students ended up running out of space and crashing their virtual disks. Support for shrinkable virtual disks has since been added to the newer versions of VMware.

The most important features lacking in VMware for deploying it in a computer lab setting are easy-to-use system administration options. VMware does not support any features to let a system administrator limit how a user configures a VM. For our course, we made the VM configuration file read-only so that students could not change it. However, students could simply create their own VM configuration files to bypass the ones we had provided. They could then change configuration settings, including access to the host machine's floppy and CD-ROM drives, the amount of memory given to the guest OS, and even the network settings. Users with root access in a guest OS could find an unused IP address and enable bridged networking to become root on the LAN. A workaround for this is to disable the bridge networking driver on the host machine. Other undocumented workarounds also exist, but they are more complicated. Simple nonoverridable configuration files would make a system administrator's life much easier. In addition to VM configuration management limitations, we found the per-user licensing requirement more tedious to administer. We would have preferred a per-machine license instead of a per-user license. Instead of having to put a license in each user's home directory, we would have preferred to simply have a license associated with each host machine.

While our use of only 16 MB of RAM per VM was less than ideal, we found that even then the performance of the VMs was reasonable. Compiling the entire Linux kernel took about 12 minutes to do when running one compilation in one VM on a given machine. It took about 22 minutes to do when running simultaneous compilations in both of the VMs on a given machine. In contrast, compiling the Linux kernel on the host OS with 128 MB of RAM took about 6 minutes. The biggest performance slowdowns when running in VMware are for operations that make heavy use of kernel code, such as during boot up and shutdown. In general, the overall performance we have observed has been quite reasonable. VMware has said that the performance is even better for newer versions of the software.

Conclusion

We found VMware to be an excellent tool for creating kernel-development environments for students to learn about operating-system design and implementation. We have since upgraded to VMware 2.0 and are using it for both the introductory and advanced operating-systems classes taught at Columbia University. We believe that VMware can be used effectively in other teaching environments where multiple operating systems, kernel-level access, and fault isolation are needed. Anecdotal evidence leads us to believe that students enjoyed their experience using Linux and VMware.