Instant-on virtual network per student: a learning environment for secure computer labs

W. Craig Scratchley, Lecturer, Simon Fraser University (wcs@sfu.ca)

***Abstract*—A low-cost, portable, lab environment for teaching topics including real-time programming, hardware interfacing, and interrupt handling is described. The environment is currently used for an embedded systems course. The environment uses VMware to create a small network connecting a host development machine with a small number of virtual target machines. The targets can be a mix of uniprocessors and dual-processor machines. The targets can take over physical hardware from the host computer in addition to having emulated hardware, allowing for a range of hardware interfacing and interrupt handling assignments one can give students. Configuration of the environment is described including techniques needed to get the environment reliably working in secure student computer labs where students have restricted permissions.**

I. Introduction

MANY academic institutions offer, through their Computing Science or Engineering units, one or more courses dealing with real-time programming and/or embedded systems. These courses often include hardware interfacing and interrupt handling as topics. Providing lab environments when these courses are offered to significant numbers of students can pose a number of challenges. Let us consider a scenario such as one we have in the School of Engineering Science [1] at Simon Fraser University. We want our students to learn to be able to write clean interrupt-handling and hardware-interfacing code that can run on an embedded device running an OS designed for real-time systems. In the School of Engineering Science, we happen to be a part of the QNX Academic Program and use the QNX Neutrino Real-time Operating System (RTOS) [2]. The issues described in this document should also apply if a different RTOS is used.

Here are some issues to consider for a lab environment for such a course:

1) The students need to run the QNX Momentics IDE and also be able to run and debug their QNX real-time programs. The IDE can run "self-hosted" on the same QNX computer where programs are debugged, but this is far less than ideal because a student may easily crash the QNX computer in the final stages of debugging interrupt-handling code. In addition, this requires the maintenance and administration of student-accessible computers running a completely different operating system. Do university system administrators have the required expertise or time? With QNX, and likely other RTOS choices, for a program to be able to read and write to hardware i/o registers and attach to hardware interrupts, the program requires root (superuser/administrator) access. For reasons including the protection of computer configuration, would we want user programs to run with root permission on such QNX installations?

2) The IDE can also run on other operating systems such as Windows XP. This requires a connected target machine on which to run and debug the real-time programs. Because students may crash targets, may set their threads to arbitrary priority levels, and may be interfacing with a particular piece of hardware on a target, having more than one student simultaneously debugging their programs on a given target should be avoided. Remember that real-time schedulers are not designed to provide fairness. A solution is that each computer running the IDE should also be connected to a target machine, and, because of the root-permission issue described above, the software installation on each target machine should be able to be reset at will. Does this increase the expense of the computer lab?

3) Many students like to do assignments on home computers or laptops. Facilitating this can reduce the needed size and expense of student computer labs. All the software I use can be licensed by students for no charge. However, will each student working on a laptop need to borrow/purchase a QNX target machine?

To provide a cost-effective, convenient, and portable lab environment with these issues in mind, I have come up with a setup based on VMware software [3]. Specifically, I have configured multiple QNX targets in multiple VMware virtual machines. In this paper, my discussion is based on VMware releases stable at the time of writing: Player version 2.0, Workstation version 6.0, and Server version 1.0.

II. THE ENVIRONMENT

In the environment I have developed, the targets use two different IP addresses on a virtual network that gets created inside each Windows XP computer on which the software has been installed and configured. Fig. 1 shows a logical view of the environment. One has a choice of launching one of two virtual machines for each IP address.

The first IP address, which corresponds with the hostname "wintermute", has one virtual machine that boots from scratch every time the virtual machine is launched. In this way the students can observe the QNX launching process and can even get into the BIOS setup utility for the virtual machine – this helps them to understand that the virtual machine is, from many points of view, indistinguishable from a real physical machine. The other wintermute virtual machine is a clone of the first but for which a snapshot has been taken after the machine was booted. Because of the snapshot, this second virtual machine launches very quickly and saves the time of the QNX booting process. When a student is crashing his target trying to get bugs out of interrupt handling code, being able to quickly restore the state of the target is very much appreciated. The other virtual machines discussed below also launch using "booted" snapshots.

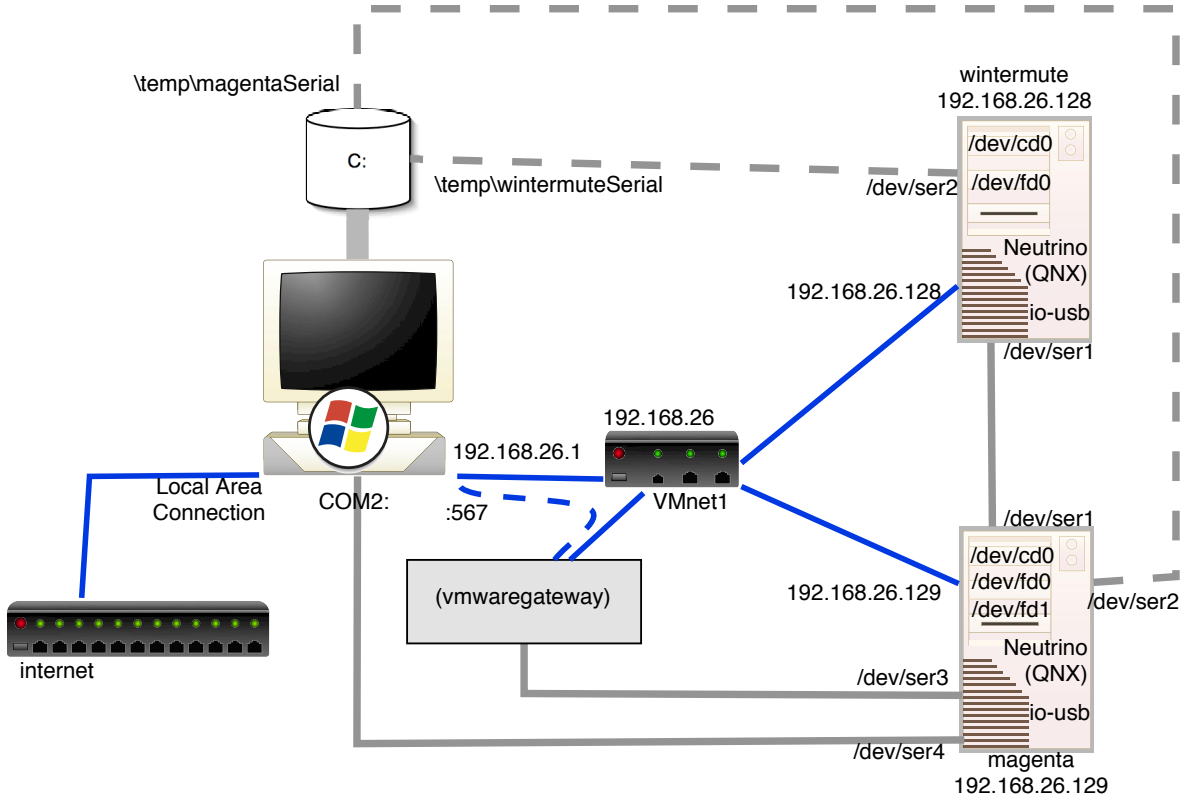

Fig. 1. A logical view of the lab environment showing the Windows XP development platform, the QNX targets, networking, and serial cabling.

The second IP address corresponds with the hostname "magenta". Since wintermute and magenta are on a virtual network (together with the host Windows XP computer), the students can easily experiment with the built-in QNX support for network-distributed operations.

Whereas the wintermute virtual machines are both uniprocessor systems, magenta has a choice of uniprocessor or dual processor virtual machines (VMware will only allow a dual processor virtual machine to be launched if the host computer has multiple processors or at least a hyperthreading processor). When demonstrating correctness issues in real-time and multi-threaded programming, being able to run the same program on both uniprocessor and dual processor systems can be very informative. Similarly, when teaching spin-locks to students in relation to Interrupt Service Routines, shared data, and multiprocessors, allowing the students to easily experiment with spin-locks (and make mistakes) greatly facilitates their learning.

The magenta virtual machines also take over some of the host computer's hardware. Specifically, the magenta virtual machines by default take over the host's floppy drive, one of the host's physical serial ports and also one of the host's parallel ports. See Fig. 2. All the virtual machines can share with the host its CD drive and simple audio output capabilities. In addition, each virtual machine can get exclusive access to a USB device like a USB drive, and each virtual machine has access to emulated serial and parallel ports, and an emulated floppy drive. (Note that, to reduce clutter, parallel ports and audio output are not shown in Fig. 1 and Fig. 2.)

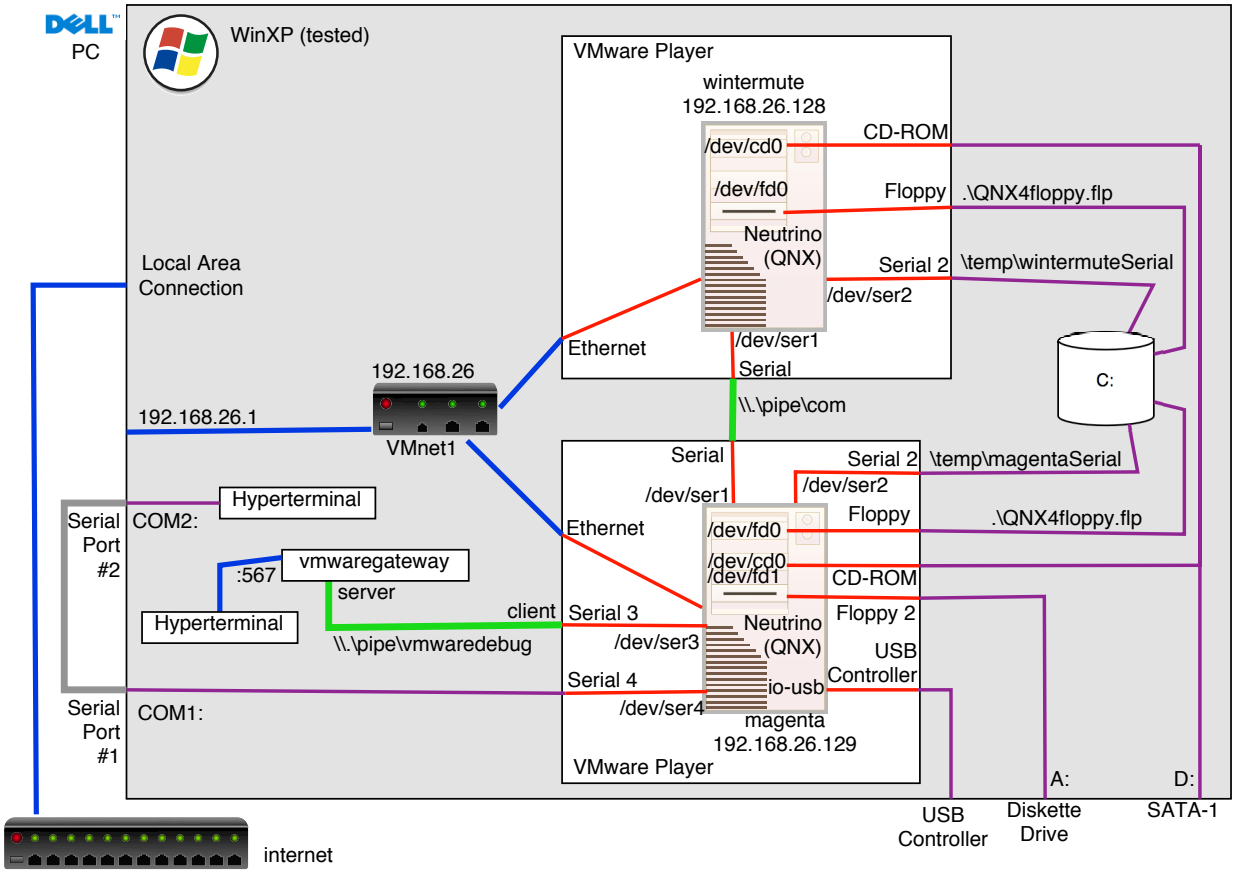

Fig. 2. The actual configuration of the software for the lab environment.

The fact that VMware allows a virtual machine to take over host hardware allows for convenient and accurate hardware-interfacing and interrupt-handling assignments. Currently, I have my students using QNX to interface with and handle interrupts from PC UART (Universal Asynchronous Receiver/Transmitter) serial hardware. The two serial ports on each lab computer are connected with a null-modem cable to facilitate testing. The students work on the serial-interfacing code for usually just one of the ports, and often use a terminal program like Hyperterminal to interact with the port on the other side of the physical null-modem cable. For students writing code on laptop computers without serial ports, they can test their code with one of the emulated serial ports (see below).

In the past I have also given an assignment in which students compare single-threaded versus multi-threaded versions of a program that accesses files on multiple devices (hard disk, floppy disk, CD, ram disk, and USB drive). Even on a uniprocessor, the multi-threaded version of the program runs in a fraction of the time it takes a single-threaded version of the program.

For distribution to my students and for use in two different computer labs, the "booted" snapshots were taken on two host machines, a machine with an older Pentium-4 Hyperthreading chip, and a machine with a Pentium-D (dual-core) chip. The snapshot from the Pentium-D chip would not work at all on computers employing many older processing chips, like the Pentium-4 Hyperthreading chips in the Dell Optiplex GX-280 computers in our Real-time Student Computer Lab. That Pentium-D snapshot, however, does work with newer chips like the Intel

Core-Duo, as long as the user clicks through a warning message similar to the one shown in Fig. 3. From my class of over 40 students, no student reported not being able to get the virtual machines running with one of the two provided sets of snapshots. For any student who didn't want to click through the warning message, or if the software is installed on a machine on which the provided snapshots do not work as desired, I provided on my course CD a set of "not-yet-booted" virtual machines all ready to be booted and "snapshotted". Each of the machines I took snapshots on was equipped with all the physical devices I had configured the magenta virtual machines to use (serial and parallel ports, floppy and CD-ROM drives, USB and audio controllers).

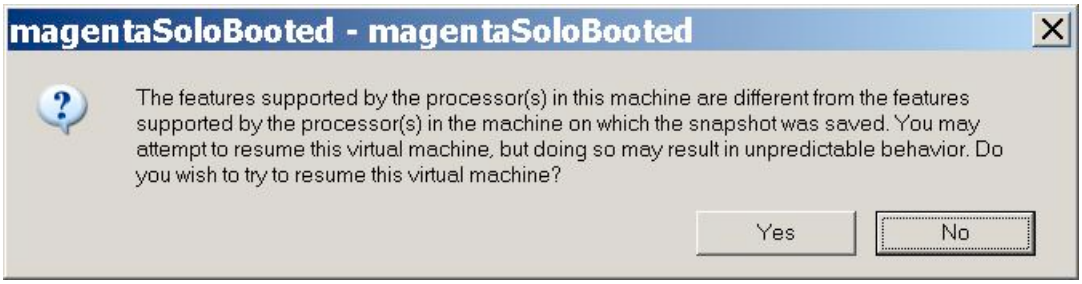

Fig. 3. Warning message often seen by students when using provided snapshots on their own computers.

When a virtual machine, even one with a snapshot, is launched, a warning message will appear for each virtual device hooked up to a non-existent device on the host computer. For example, the VMware "Floppy 2" device is hooked up to a physical host floppy drive for the virtual machines in which magenta runs. If the host doesn't have a floppy drive, a warning message is displayed. To avoid the warning messages, one can simply "disable" the virtual device in the VMware .vmx configuration file for the virtual machine. As examples of this, I provide to my students .vmx files like magentaSoloBooted-laptop.vmx which disable the floppy drive and serial and parallel ports normally connected to physical devices.

Through using the "linked cloning" feature of VMware, the bulk of the disk image space for the virtual machines resides in a single file in the directory of the first wintermute virtual machine. This saves considerable disk space and, presumably, host computer memory. As for disk space, I have been able to fit for the students on a single CD all the software for my course, including QNX Momentics, VMware Player, the virtual machines, and other software my students will use such as MinGW, various plugins for Eclipse (which forms the basis of QNX Momentics), pdf books and other documentation, and a number of miscellaneous development tools.

Because by the time I take the snapshots the running virtual machines have already acquired their IP addresses, it is necessary that all computers using the snapshots have their VMware virtual network configured for the same subnet. For my configuration, the subnet ended up being "192.168.26". The VMware installer seems to randomly assign a subnet when installing the VMware virtual networking. For this reason, both student-lab administrators and students will need to run the program "C:\Program Files\VMware\VMware Player\vmnetcfg.exe" after installing VMware and change the subnet so that it corresponds with the subnet that the snapshots are using. When installed on Microsoft Vista, vmnetcfg.exe will sometimes have to be explicitly "Run as administrator" via a contextual menu. In addition to allowing "booted" snapshots to be used, having a common subnet has the added advantage that I can use a set of simple instructions common to all computers running the course software when describing how to connect to the target machines from the QNX Momentics IDE (e.g. the magenta IP address is always 192.168.26.129).

III. Serial ports and vmwaregateway.exe

As mentioned in section II, the computers in the student computer labs have two serial ports connected together by a null-modem cable. For students working on a home computer with only a single built-in serial port, in theory they should be able to purchase a USB serial dongle (advertised for as low as about $10.00) and connect the serial port and serial dongle together with a null-modem cable. I suspect they would need to have the Windows terminal program like HyperTerminal connect to the serial dongle end. For students wanting to work on a laptop or home computer without any built-in serial port, or for any student willing to interface to an emulated UART chip, magenta's /dev/ser3 serial port uses VMware's serial port via named-pipe feature. I wasn't able to find a Windows terminal program that could communicate with a named pipe, but I did find the third-party opensource program vmwaregateway.exe [4] that listens on TCP/IP port 567 on the local Windows host and forms a gateway between

that port and the named pipe \\.\pipe\vmwaredebug. Since many Windows terminal programs including HyperTerminal allow TCP/IP connections, I was able to connect programs like HyperTerminal through to magenta's /dev/ser3 on any host computer, even if lacking a built-in serial port. Logically, port 567 looks like a tunnel to some sort of TCP-to-serial appliance on the network (see Fig. 1).

There does seem to be, however, a problem when using the version[1] of vmwaregateway.exe I have obtained: some temporary problem occurs when the character 0xFF is transmitted though the program. I have not had time to investigate in more detail.

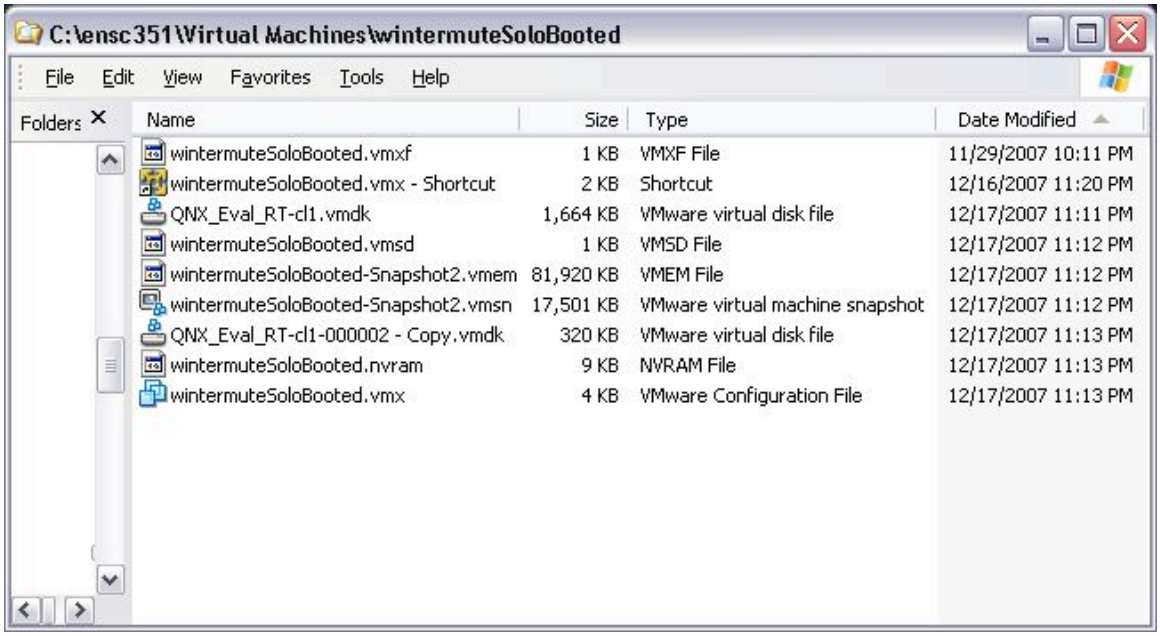

Fig. 4. Permanent read-only files in the directory "wintermuteSoloBooted".

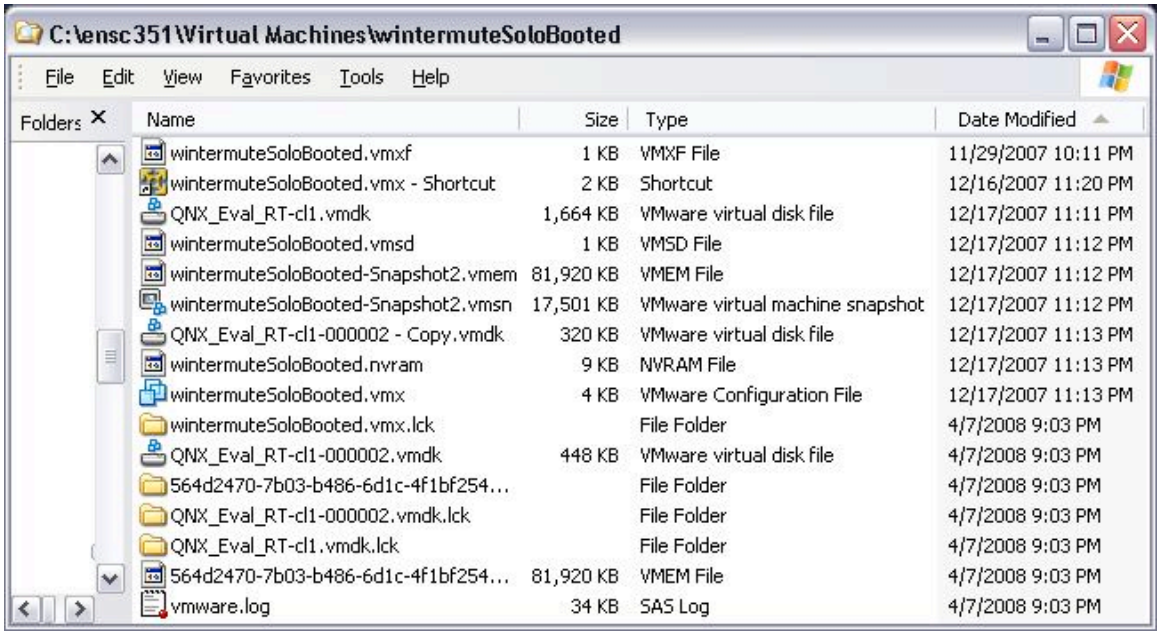

Fig. 5. Files and directories in the directory "wintermuteSoloBooted" while the virtual machine is running. The files dated 4/7/2008 are not needed after the virtual machine is closed (QNX_Eval_RT-cl1-000002.vmdk will be replaced before the virtual machine is launched again).

[1] un-numbered, 32,768 bytes in size, 16-Nov-2000 – 11:15 provided as date and time.

IV. Deployment techniques for student labs

Like for many software packages I find useful for my students, certain techniques should be used when configuring VMware and vmwaregateway for use in secured student computer labs where students have accounts with restricted permissions. Even Windows itself requires that certain techniques be used for the configuration.

vmwaregateway.exe can most conveniently be used by installing it as a Windows Service. Such services can be configured to start automatically when the Windows computer is started. Unfortunately, when students login to Windows with their restricted accounts, vmwaregateway.exe does not function properly when it has been started as a service. To get around this problem, I have the students launch the program in test mode via a shortcut. vmwaregateway.exe is an open-source project, so someone with the requisite knowledge should be able to fix this.

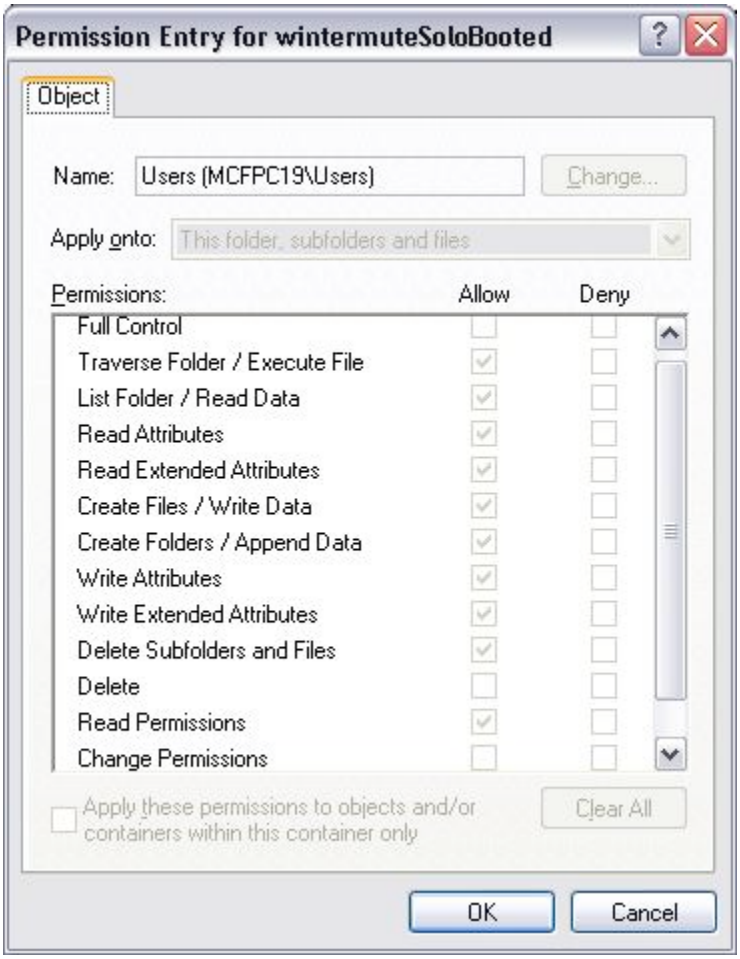

Fig. 6. Special student permissions for the directory "wintermuteSoloBooted" containing files for a virtual machine.

When wanting to have a linked-clone virtual machine with a snapshot, even when one specifies that the virtual machine should revert to the (last) snapshot when powered down or rebooted, VMware always requires write access to one of the files used for the hard disk image. In addition, VMware creates additional temporary files and directories in the directory for the virtual machine. I was unable to have these temporary files and directories created in a temporary directory like C:\tmp. Given that the virtual machines including the files for images are on a portion of the C: drives where students do not normally have write access, I have had to do such things as design a set of NTFS permissions so that VMware can do what it needs while still keeping all needed non-transient files read-only. Fig. 4 shows the directory for the WintermuteSoloBooted virtual machine as it might appear before the virtual machine is launched. Fig. 6 shows the special permissions for this directory for student accounts. Note that files and subfolders can be created and then deleted. Fig. 5 shows the directory listing while a virtual machine is running. When the virtual machine is closed, most of these files are automatically deleted. In order to protect

important files from being modified or deleted, the files shown in Fig. 4 had their permissions for student accounts modified as shown in Fig. 7. Specifically, student users are denied write permission for these important files.

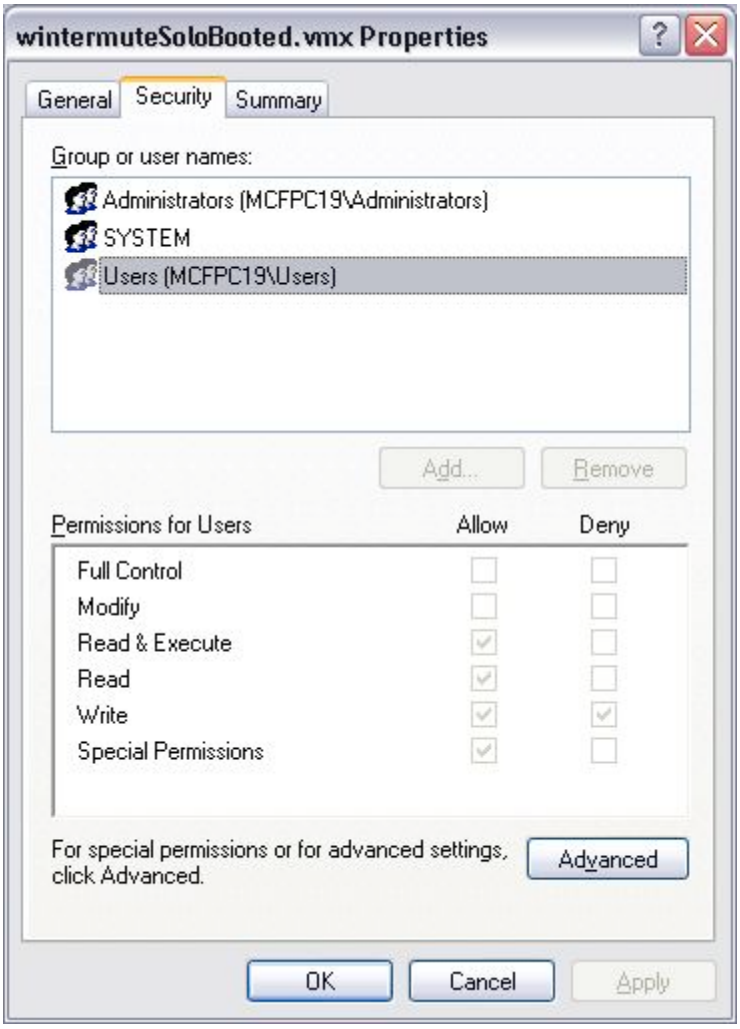

Fig. 7. Student users are denied write permission for important files in the directories holding virtual machines.

For each virtual machine, I configured a shortcut to replace from a read-only backup a small .vmdk file before launching the VMware application (the shortcuts are discussed further at the top of the next page). Replacing the file is not optional. With the other files read-only as I have configured them, not doing the replacement results in VMware seeing parts of the hard disk image corrupted the next time the virtual machine is launched.

I had the choice of three VMware programs for my students to use for the course: VMware Player, VMware Workstation, and VMware Server. VMware Player can be used with no charge and does not even require a license key. Students can have access to VMware Workstation for no charge through the VMware Academic Program. A VMware Server license can be obtained for no charge through the VMware Academic Program and also by registering on the VMware website. I chose VMware Player for my students to use because it is the smallest in terms of size and also has the simplest and cleanest user interface.

For the student environment to immediately work as intended, VMware Player needs modified default values for application preferences. The most important of these is that exiting a window of the VMware Player application should "Power off" the virtual machine that was displayed in the window instead of the "factory" default of "Suspend" the virtual machine. Because of settings in the virtual machines themselves, powering off a virtual machine will result in the machine's intended snapshot being loaded the next time the virtual machine is launched. To enable for students the use of the modified default values on the University's lab computers, we set the preferences as we wanted them and copied the file "%appdata%\vmware\preferences.ini" to the Windows default

profile used as a template for the profile created when a student logs into a lab computer. For home use by students, a configuration batch file simply copied *preferences.ini* to the student's profile on their home computer.

VMware Player can be downloaded and easily installed stand-alone, but it also comes bundled in the VMware Workstation package. In our real-time student computer lab it is the VMware Workstation package that I have installed. I didn't want the VMware warning of Fig. 3 when launching a virtual machine with a "booted" snapshot, so wanted to be able to take snapshots in the lab, and VMware Player can't take snapshots. The VMware Server program can also make snapshots of machines with the features I need for my course, but VMware Server cannot co-exist on a host computer with VMware Player.

Having installed VMware Workstation we ran into an issue with Windows XP. When the students logged in, I wanted .vmx files, which provide the configuration for VMware virtual machines, to be opened by default by the VMware Player application. However, there seemed to be no way to set Windows up ahead-of-time to make this default association for students. To solve this problem, I put the full pathname of the VMware Player application into each shortcut used to launch a virtual machine. In order to allow the same shortcut to be used irrespective of which VMware package has been installed, I requested installation of VMware Workstation into the directory "C:\Program Files\VMware\VMware Player". Thus the complete target line in the wintermuteSoloBooted shortcut was set to:

```
C:\WINDOWS\system32\CMD.EXE /c copy
    "QNX_Eval_RT-cl1-000002 - Copy.vmdk" "QNX_Eval_RT-cl1-000002.vmdk" /Y && erase
    c:\temp\WintermuteSerial.txt && start "wintermuteSoloBooted" "C:\Program Files\VMware\VMware
    Player\vmplayer.exe" wintermuteSoloBooted.vmx
```

To streamline virtual machine startup, the shortcut also deletes if present the file in the directory "C:\temp" that holds output for VMware's "Serial 2" virtual serial port (see Fig. 2). Other custom configuration to streamline virtual machine startup included tricking VMware to ignore the situation that the virtual machine may be running on a different computer from where the virtual machine had been created. This was done by adding the line 'uuid.action = "keep" ' to the .vmx configuration files. Normally in such a situation VMware Player may ask if the virtual machine had been copied or moved. For this student environment, students should answer "moved", but VMware suggests answering "copied" if the user is unsure.

V. Microsoft Windows Vista

All our student computer labs use Windows XP. This appears to be a good thing. While current VMware software supports Windows Vista, I have found differences in behavior. For example, the choice of an automatically chosen physical serial port to bind to a virtual serial port is at least sometimes different when comparing execution on Windows XP versus Windows Vista.

VI. Further Applicability of such Environments

Though I have not used it for this purpose, I imagine that such VMware deployments can also be very useful when one is trying to teach distributed systems of modest size, especially fault-tolerant or real-time ones, or OS kernel design and development.

The student computer labs in the School of Engineering Science do not use Linux, but the key software packages I use on Windows XP for my Real-time and Embedded Systems course (VMware and QNX Momentics) are available for Linux. Therefore, there is a good chance that the environment described in this paper could be ported to Linux, and in fact it is possible that some of the factors I have needed to address for deployment to Windows XP as described in section IV would either not exist with Linux or would be easier to address.

References

[1] http://www.ensc.sfu.ca -- accessed April 2008.
[2] http://www.qnx.com -- accessed April 2008.
[3] http://www.vmware.com/ -- accessed April 2008.
[4] http://l4ka.org/tools/vmwaregateway.php -- accessed April 2008.
[5] http://pubs.vmware.com/ws6_ace2/ws/ws_devices_parallel.html -- accessed April 2008.