OpenVMS Technical Journal V13

Adding Physical CD support to the SIMH VAX

Michael D. Duffy

Introduction

As aging hardware keeps chugging along, maintenance costs continue to rise and replacement parts become more and more scarce. In some cases, that ancient MicroVAX is still running a critical application that has never been ported to another platform. But since the VAX never crashes and always comes back up after a power failure, nobody seems too concerned that everyone who understood the application left the company years ago. In the back of your mind, you know you should really do something about that.

Some sites choose to keep the application intact but move it to an emulated VAX running on some other system, usually at a substantial performance increase and cost reduction. Commercial emulators can be quite expensive, but open source solutions may not have all the features required to get the most out of the product.

In response to these concerns, Migration Specialties asked me if a copy of OpenVMS running under SIMH could be allowed direct access of a CDROM device on the host computer, whether it was running OpenVMS or Windows. (UNIX systems need no such enhancement, as pointing a container file specification to /dev/cdrom works with the standard SIMH version.)

SIMH is a hardware simulator that can act as any one of more than twenty different machines, but I was asked to concentrate on the VAX 3900 simulator, as it was the most likely existing configuration to be used by the customer sites we envisioned.

This article explores the challenges and solutions found by the author while adding Host CD support to the SIMH open source VAX emulator. The solution demonstrates that a developer may not need a wealth of experience with SIMH in order to add significant functionality to it, and may serve as a template for your own enhancement ideas.

The Plan

SIMH disk I/O is based on container files representing disk drives, a standard approach among many different emulators. A file on a host device acts as a repository for a bit-for-bit copy of the entire contents of a disk volume attached to the computer to be simulated. Previously, a user could create a disk image file from the contents of a physical CD and then attach the image file to the simulator. Our goal was to eliminate this time-consuming step and allow access directly to the host CD drive.

The problems I anticipated at the outset were the lengthy delays inherent in CD access and how to support OpenVMS mount verification and CD volume switching between the emulated environment and the host.

SIMH disk I/O is synchronous, that is, the simulator stops and waits for container file reads and writes before continuing. This is true whether or not the data are immediately delivered to the guest OS. Some experimentation would be necessary to determine how tolerant OpenVMS would be of these delays, and whether an asynchronous delivery mechanism would be needed.

Experimentation with CD volume changes and guest MOUNT and DISMOUNT activities would be needed to determine 1) whether there is a simple, reliable way to tell within SIMH itself when the guest has mounted or dismounted a volume and 2) whether allowing a host I/O to fail is sufficient to trigger OpenVMS mount verification on the guest side.

Proof of Concept

I decided to put together the simplest solution to probe these areas. Intercepting what SIMH thought would be a container file read and redirecting it to a CD device seemed like the obvious choice. After reading the data, I would insert it into the same buffer that would have been used for container file data and allow SIMH to proceed from there. How hard could it be?

The first step was to determine where the I/O should be intercepted and how to determine which of the I/Os passing through that point were the right ones to redirect. Routine sim_fread() in source file SIM_FIO was chosen because all container file reads pass through it. Since writes were to be disallowed, placing the test in the read dispatcher would automatically prevent writes from being attempted by any new code I would write. Later testing indicated an improvement might be made possible by moving the test elsewhere, which will be discussed later.

One of the arguments to sim_fread() is a unique file identifier, called fptr, that could be used to identify the I/Os of interest. I quickly decided that for the proof of concept, I would open a container file as normal when SIMH was initialized, but intercept the I/Os to that file at runtime and send them to the CD instead. Later, after proving the concept, I would remove the container file or at least make it into a tiny stub file that did not consume space on the host disk.

The VAX implementation in SIMH uses some of the same code as the PDP11 for I/O. Specifically the RQ* controllers and disk types are shared via the PDP11_RQ* code found in SIMH's PDP11 source directory. I looked through the code defining device types and the data structures that represent them in SIMH, finding that due to the number of bits representing the device type and the number of devices already supported, there were no available slots for adding a new device type. I therefore used the slot that had previously been taken by the RA70 device type, the last device in the list, and also one of the smaller disk models. I renamed that device PHYCD and copied the remaining values from the RRD40 device definition already present in SIMH. I then created a Physical CD Control Block to store various information related to the PHYCD device.

The next step was to modify the SET and ATTACH commands to recognize references to the PHYCD type. SET and ATTACH are used when SIMH is started, to define the devices that will be present on the simulated system and associate them with container files, respectively. I modified the ATTACH code to open a stub container file and store its file pointer in the PHYCD control block, so that Physical CD I/Os passing through sim_fread() could be identified via the fptr argument. I also changed ATTACH to treat the container file argument as a device specification when the device type is PHYCD. The new syntax is compared with the existing method below.

Traditional container file examples:

```
SET RQ1 RRD40
ATTACH RQ1 DKA0:[SIMH]CD_IMAGE_FILE.DAT
```

New Physical CD access examples:

```
SET RQ1 PHYCD
ATTACH RQ1 DKB400:  (OpenVMS host)
```

This example causes SIMH device RQ1 to be attached to physical CD device DKB400: on the host system.

Now it was time to actually redirect an I/O to the CD drive once it was detected, but the information passed to sim_fread was still incomplete: The desired length of the read is present, but the starting position is not. SIMH sets the container file read offset via an fseek() before sim_fread() is called, so I added a similar test at the fseek() to record the desired starting position in the Physical CD control block.

Since the guest OpenVMS always starts reads on 512-byte boundaries, I did not have to make any adjustments to the starting position or length of read on an OpenVMS host. On a Windows host, however, reads start on 2048-byte boundaries, so some calculations must be performed to read the proper block and extract the portion the OpenVMS guest requested (and introduce an opportunity for a small read-ahead cache, since the data would otherwise be wasted.)

These values were passed to a $QIOW to read the CD and place the data into the buffer ordinarily used for the container file.

For the purposes of initial testing, I added a $MOUNT system service at SIMH startup to ensure the CD device would be available. For now, no provision for removing or switching CDs was included. That would come later.

Initial Testing

Once this bare-bones approach was ready, I booted up a copy of OpenVMS/VAX V7.3 with the PHYCD unit attached to device DUA1: and placed a CD in the host drive.

As I anticipated, there were delays of a few seconds whenever the CD needed to spin up, but the guest copy of OpenVMS was very tolerant of these delays, with no failures or errors noticed to date. A given application might be less forgiving, but OpenVMS itself seems not to mind very much.

Once the CD was up to speed, further reads proceed normally, with no effect on the simulated system that would be obvious to a human. However, overall system throughput is reduced during the time the CD is being actively read. This is also true of container files, but is magnified somewhat by the lower performance of CD drives.

It was decided to expand this concept to include support for switching CD volumes, handling errors and triggering OpenVMS mount verification within the guest OS at the right times.

Supporting CD Swapping and Removal (OpenVMS)

My initial thoughts were that the OpenVMS and Windows-host version would differ greatly with regard to handling the user removing or switching the CD in the drive. The two versions turned out to resemble one another much more than I originally thought, due to an error I made early in development and didn't catch until a significant amount of code had been written. The reader can use my mistake to avoid doing extra work on any similar project.

My first attempt at mount verification support on OpenVMS hosts was based on this general idea: I planned to detect mount verification on the host and simply pass it into the emulated environment. First, I would issue the I/O with a timeout event flag. If the I/O timed out, I would check to see if the host CD is in mount verification. If so, I would cancel the I/O, perform an internal DISMOUNT/ABORT and Mount the CD again. Then I would reissue the I/O, while simultaneously notifying the guest operating system that mount verification should be triggered.

It was at this last step where I made a slight mistake by writing all the other code first before carefully examining the mechanism by which I would notify the guest that something unusual had happened. As it turned out, the code worked well, but by the time it was triggered, it was already too late to notify the guest OS.

Had I checked in a little more depth, I would have found that the opportunity to signal mount verification (which means to return an offline status for the emulated device) had already passed. Routine rq_rw_valid(), a series of validity checks that gets performed before the I/O is started, is the right place to do this, but is already finished before the I/O is sent on its way.

In order to get the solution working more quickly, I made the OpenVMS version behave more like the Windows version (already underway), but left the aforementioned code in place with an eye toward fixing it at a later date. I envision doing the read preemptively in rq_rw_valid() and leaving the data in a buffer that can be retrieved later in sim_fread(). In this way, timely mount verification alerts can happen, while avoiding extra steps.

Supporting CD Swapping and Removal (Windows)

How, then, to support OpenVMS-style mount verification on other than an OpenVMS host? Some mechanism was required to detect when a different CD appeared in the drive. I decided to copy the first 2KB of the drive "from time to time," the exact meaning of which had yet to be determined. Then, when a different CD appeared in the drive, a different 2K of data would be found in the first block, and SIMH could return "offline" to the client while simultaneously updating the cached copy of the new data. SIMH could then continue normally, while the guest OS began its mount verification processing.

In order to have the volume checked on a regular basis, I decided to close the handle to the host drive after a few seconds of inactivity. Any period of time long enough to physically switch the CD would trigger this condition, after which SIMH would detect the new CD.

Regularly opening and closing the handle to the drive does add some overhead, but the delay happens at the same time the CD is coming up to speed, and so only slightly lengthens an already noticeable delay.

Once this mechanism was working, I modified the OpenVMS version to do the equivalent opening and closing via $DISMOU and $MOUNT. It works as well for OpenVMS as it does for Windows, but the OpenVMS version also contains the code described in the OpenVMS-specific section, which a future version can take advantage of.

Both the OpenVMS and Windows host version also return "offline" if the CD drive contains no CD, causing the guest OS to behave as expected. OpenVMS returns "medium is offline" if a MOUNT is attempted, or retries once per second when mount verification is underway.

Room for Improvement

The code described above represents the first attempt at supporting physical CD access within SIMH. There are a couple of obvious improvements that can and will be made as time allows.

First, the inactivity timer test is currently located in the VAX CPU instruction dispatcher. This was a convenient and reliable place to put it during testing, but is not ideal. Ideally, a standard SIMH timer event should be used, which will reduce overhead.

Secondly, the OpenVMS version could benefit from placing a read into rq_rw_valid() for the reasons described above.

Finally, the code as currently designed sometimes triggers an "offline" signal when it is not necessary to do so. Any time a CD volume is switched (and also when SIMH is started with an empty CD tray), the "First-2K test" will trigger. Mainly, this is because I am currently unaware of a good way to tell when the OpenVMS client has dismounted a volume. From SIMH's perspective, I/O simply stops for some period of time and resumes later. If SIMH could reliably know when a DISMOUNT has occurred, First-2K testing could be suppressed for the next sequence of I/Os.

Possible ways to improve this behavior include detecting a guest MOUNT, perhaps by recognizing the specific block number(s) requested, or by detecting a DISMOUNT/UNLOAD by some condition recognizable from the PDP11_RQ code. Currently, a spurious "medium is offline" or transient mount verification cycle remain as the only obvious bug in the new functionality. It should be noted that the guest OS can simply retry the operation and everything will work itself out, but it would be preferable to remove this behavior.

For more information

The SIMH Website

http://simh.trailing-edge.com/

There you will find the SIMH base source code and many resources.

The SIMH mailing list

Send a SUBSCRIBE message to simh@trailing-edge.com

The Migration Specialties website

http://www.MigrationSpecialties.com/

Source code and installation kits for the enhanced SIMH functionality