

Short History of IBM's Virtual Machines

There are several uses of the term "virtual machine". In general they seem to describe a program that behaves somehow as a machine. The "Java Virtual Machine" is such a usage. I want to tell here about a narrower earlier kind of virtual machine that was once better known among programmers.

When IBM introduced the 360 model 67, circa 1967, it was IBM's first major system with virtual memory. When the machine was announced a software system called TSS was promised. Perhaps this was a reaction to the earlier [Multics](#) announcement. TSS was a substantial departure from IBM's other systems and was slow to be delivered. A team of IBM people in Cambridge MA conceived and built a program for the 67 that would provide the illusion of several standard 360's which lacked virtual memory. This program was called CP 67. The Cambridge team also produced a simple interactive one user operating system called CMS. CMS would run happily and efficiently on one of the virtual machines provided by CP 67, or on a real machine for that matter. Tom Van Vleck provides very interesting [history and references](#). In fact he provides much of the information that I had intended to write here. I will repeat little of that information here.

As of several years ago, VM/370, a direct descendant of CP 67, was used about equally by users of CMS and those who needed to run various alternate operating systems for reasons such as testing.

Today a rather direct descendant of CMS runs on VM/390. IBM has had a policy for about two decades now of phasing out VM. It is yet heavily used inside and outside of IBM. Ironically the developers of competing operating system within IBM use CP/390 very heavily to test and debug their systems. IBM now (1999) recommends VM/390 for Y2K testing.

[VMware](#) provides a similar system for Intel 32 bit machines. Many of the points of their FAQ make sense in light of virtual machine strategies. [Xen](#) from Cambridge University is a similar open source system.

Just What is a Virtual Machine?

A virtual machine is the construct of a program (such as CP/370) that behaves so much like a real machine that an OS, or other program written to run alone on a real machine, is fooled into thinking that it is running on a real bare machine by itself!

One way to do this is to simulate the hardware's appearance to the software. The commercial products [RealPC](#) and [Virtual PC](#) run on a Mac and simulate an IBM compatible PC. Those programs must simulate each x86 instruction just as it would be executed by x86 hardware.

There is a more efficient way if the machine that you would simulate is the same as the machine that you run on. If you have a zot why would you want to simulate a zot? Here are a few reasons:

- Provide interactive debugging of privileged code, such as an operating system. On expensive machines it may be difficult to find time for such activity. The virtual machine may have an operators console thru which to examine the machine state. Current real machines seldom have that. For instance you may be able to "single step" privileged programs on a virtual machine.
- Test machine architectures with altered privileged mode before they are built. Debug code written against the new features,
- Test new versions of operating systems before deployment.
- Run distinct versions of operating systems simultaneously.
- Test for Y2K compliance by setting the clock of a virtual machine ahead.
- Time share for interactive development of applications either to run in production or in decision support modes. Such programs can be granted read-only access to real mission critical data.
- Create separate compartments for classified information processing. The isolation of virtual machines is simpler to understand and depends on less code than most alternatives.
- Run programs that require hardware that is virtually available but not really available, such as very large RAM.
- Simultaneous combinations of the above.

Since the early 60's machines have generally had features, such as [a privilege mode](#) which, together with a memory map, allows an operating system to impose discipline, and thus protect itself and protect one guest from another. We will refer to such an OS, as the "CP" (Control Program).

CP can use this facility to confine the effects of running guest programs in the real machine to those parts of the hardware that CP explicitly allocates to those guest programs. The guest program can run on the bare hardware without the simulation of each instruction. The guest code runs in "problem mode" which is the alternative to privileged mode.

The 360 and 370 architectures adhered to a discipline that made virtual machines possible. This architecture was set by people who had presumably not yet thought or heard of virtual machines. Many other systems of that day or this, did not adhere to these unexpressed principles.

The best I can do to explain those principles is to say that the hardware should be designed so as to allow the privileged code to completely hide its decisions on just how to provide the real machine illusion.

Wang corporation produced machines similar in many ways to the 360 but failed to

duplicate this property and a VM could not be written for a standard Wang machine. Many models of Intel's machines allow user code to read registers and get the value that the privileged code put there instead of the value that the privileged code wishes the user code to see. The privileged code must relate to the user code just as the hardware relates to the privileged code. The hardware must determine what the virtually privileged code sees as it reads a privileged register. This provides the user code with an environment that is identical to what it would be if it were in running on a real machine instead of a virtual machine. More on this later.

A real machine runs in [one of two states](#), and so must a VM. When the VM is in privileged state (virtual privileged state) the real machine is in problem state in order that the actions of the program not damage CP or other virtual machines. When a privileged instruction is attempted the real machine traps so that CP may interpret that instruction relative to the state of the VM. Each VM has a complete set of privileged registers implemented by the CP. When a VM runs, CP will load some of the real privileged registers with those from the VM but other real special registers will be loaded with values that are designed to fool the guest program by providing a complete machine illusion.

Thinking about virtual machines and their logic broaches many architectural issues.

I will write the following assuming that the reader is familiar with privileged mode architecture. Maybe a better intro later.

CP's plan is to run all code other than itself in user mode, just as would a classic operating system. Following IBM we call that other code the **guest code**. The code in the machine is thus divided into CP and several guests, one per virtual machine. A part of each guest program will be code that was designed to run in privileged mode and when such a part runs, it must produce the right effects relative to its own virtual machine. We call these privileged parts VP (for Virtually Privileged). Most of the instructions in VP code are unprivileged and the trick is to run VP code in user mode. When a privileged instruction is encountered, the real machine traps to CP which can maintain the illusion by faithfully interpreting the instruction, but relative to those resources actually devoted to that virtual machine.

Each virtual machine has, at any instant, some set of real RAM pages devoted to it. CP builds a memory map that allows the guest code to run in that RAM which then appears to the guest code as real RAM. The RAM devoted to a virtual machine would vary from moment to moment but invisibly to the guest.

When the VP code tries to read or write the disk the involved privileged code traps and CP knows what portion of some real disk is devoted to playing the role of the virtual disk. Real IO performed by CP plays the role of the virtual IO. The real IO may run concurrently with the virtual machine. The guest does not know the

difference. There was once a [serious, but instructive bug](#) in the VM's IO logic.

Within VM/370 disks were allocated to a virtual machine thru administrative actions. These virtual disks (called mini-disks in VM/370) provided long term storage.

Virtual machines had and used virtual card readers and punches. These could be mapped onto real readers and punches but this was seldom done. A virtual machine might have several decks "at hand" which could be placed in the card reader or passed to other virtual machines. Such IO served many of the purposes that Unix puts pipes to.

Printers (600 line per minute mechanical and noisy) were virtual and often mapped to a real printer via print queues.

VM370 [solved a class of security problems](#) that were difficult in traditional systems.

[Here](#) is how VM370 provided virtual machine that were equipped with virtual memory.

To be integrated:

Perhaps IBM pioneered the virtual machine when they built CP67 in about 1967 which ran on the 360 model 67 which was IBM's first machine with virtual memory. CP67 was a privileged mode program that provided multiple virtual machines, each very much like a real machine. The simulation sufficed to run IBM's other operating systems that were designed for the 360. A successor, VM370, was even able to run itself. Privileged mode instructions were interpreted but user mode instructions (problem state) were executed directly by the hardware and thus the performance of virtual machines was usually excellent. Virtual machines could be configured to communicate with each other just as could real machines. The configuration of virtual machines was done by editing simple files stored outside the purview of the virtual machines. One could examine such files and easily see what connections there were. Indeed this was easier than exploring the tangle of cables typical of a computer room to find the connections between real machines.

This virtual isolation was used many ways, such as testing new releases of operating systems, debugging privileged code, and sometimes to provide the isolation required by special security needs, all of this while the real machine continued mission critical work within another virtual machine.

IBM's current [VM offerings](#)

Web [servers](#) using VM

[Comments](#) from Jeffrey Savit

Melinda Varian's [musings](#) and many links (See her 'VM and the VM Community')

pub.)

More insight and pointers may be found in [Tom Van Vleck's note](#).