

Kevin Lawton, <http://plex86.sourceforge.net/>

## ***The new Plex86 x86 Virtual Machine Project***

**W**elcome to the new plex86 project. Plex86 has been rehashed/revitalized to offer a very lightweight Virtual Machine (VM) for running Linux/x86. Rather than implement a full and heavyweight VM which can run all guest Operating Systems (OSes), the new approach is designed to run only Linux VMs, making the new plex86 architecture on the order of 10x or 100x more simplistic.

**A**s Linux does not make use of "corner cases" which necessitate complex software techniques (e.g. dynamic translation and other execution monitoring methods), plex86 now implements no such techniques. Rather, guest Linux code is executed as-is in a safe VM environment managed by a VM monitor. Guest kernel code is executed at a less privileged level, causing privileged operations to generate exceptions which are managed by the VM monitor.

**V**irtualized IO model: because the Linux kernel can easily be configured before compiling to remove support for most all IO hardware, the model for handling IO in the guest Linux VM is greatly simplified and improved in plex86. Excepting for some very lightweight emulation of a few core components (e.g. interrupt controller, system timer), plex86 does not model(emulate) IO hardware. All IO interactions occur through a Hardware Abstraction Layer (HAL) which passes packets of information between the guest Linux and the host OS. This not only removes the complexity/overhead of modeling IO hardware, but allows for more flexibility in dealing with guest-host interactions and creates a truly virtualized Linux resource which is unrooted from the actual hardware. Guest Linux drivers interact with the HAL to manage IO related interactions (networking/disk/etc) with the host OS.

Q: What is this new plex86?

A: It is a very lightweight x86 virtualization strategy which executes x86 code as-is, but inside a virtualization container. It does not use any dynamic translation nor dynamic code scanning techniques to deal with unvirtualizable x86 instructions. But rather executes only code which is virtualizable by nature. As well, this new architecture does not model the IO hardware.

The x86 ISA (Instruction Set Architecture) is not 100% virtualizable, especially with respect to system code. To build an x86 VM which can execute arbitrary binary-only OSes and related software on current x86 hardware requires a lot of software complexity to overcome the unvirtualizability of the x86 ISA. And would also require that reasonably complete IO modeling be implemented, as the guest OS executing within the VM needs to drive a distinct set of hardware which can not conflict with the host hardware.

But Linux (and perhaps other OSes) require extremely few modifications to make it execute within the plex86 VM, and can be configured easily to trim out unnecessary IO hardware and CPU support which would otherwise require a heavy VM. What remains is a very lightweight and highly virtualized Linux VM resource which is completely untied from the host hardware. And in fact, rather than model IO hardware for functions such as networking/disk/console, special Linux guest drivers communicate packets from the guest to the host via a HAL (Hardware Abstraction Layer). This makes the VM very simplistic and abstracted from IO hardware. It can also be more efficient, as no IO emulation is necessary.

One of the ways that plex86 achieves this simple strategy is that it executes kernel code at user privilege where many system instructions will naturally generate exceptions and can be monitored. The modifications to Linux necessary to make it VM'able within plex86 actually only affect the Makefiles, adding an extra compile option which forces the inclusion of an assembly macro file. This file redefines the meaning of a few instructions, notably PUSHF/POPF, because their behaviour with respect to the interrupt flag is broken.

Another use of this lightweight style x86 virtualization is to accelerate an x86 emulator (like bochs), for those parts of execution (for example application code) where heavy virtualization is not necessary. Execution of system code and IO can be left to the emulator to handle.

Q: Why do I have to recompile Kernel to run it on a Virtual Machine ?

A: Well, first to compile out all the non modularized IO. But this is only a matter of running 'make xconfig' or your preferred kernel configuration method. Having IO expectations is antithetical to this kind of VM. IO should be communicated between the guest and host via the HAL, and thus clean communications without any need for IO emulation can occur and the VM can be kept very lightweight. Second, essentially because Intel broke the semantics for handling the interrupt flags (EFLAGS.IF) for a few instructions (like PUSHF/POPF) while using PVI (protected mode virtual interrupts). PVI is the protected mode counterpart to VME for v8086 mode. It lets application privilege code manipulate the interrupt flag without generating as many exceptions for higher performance processing, and is great for just such a thing as virtualization. However, strangely enough, while it works well for the STI/CLI instructions, PUSFH/POPF/IRET have broken semantics even though they are implemented properly for VME and the same semantics are what are needed. Go figure. Anyways, a couple macros do the trick, expanding out to a few instructions which implement the correct semantics.