Virtual 8086 Mode

From OSDev Wiki

Virtual 8086 mode is a sub-mode of Protected mode. In short, Virtual 8086 mode is whereby the CPU (in protected mode) is running a "Emulated" 16bit Real Mode machine.

Contents

V86 Problem

The most common problem with v86 mode is that you can't enter Protected mode from inside of a v86 task. In other words, if you are running Windows or have emm386 in memory, you can't do a "raw" switch into protected mode (it causes an exception). DOS extenders worked around that problem using either VCPI or DPMI interfaces to switch into pmode (actually, promoting their V86 task as a 'regular' user task). For an OS programmer such interfaces are simply useless as they're part of another OS.

There are a few other more "technical" problems people have when using v86 mode, mostly because v86 has some instructions "emulated" by what's known as a v86-monitor program, as the CPU is in protected mode, some instructions are high up on the security/protection level and running those directly would cause no-end of trouble for the OS.

Detecting v8086

EFLAGS.VM is NEVER pushed onto the stack if the V86 task uses PUSHFD. You should check if CR0.PE=1 and then assume it's V86 if that bit is set.

```
detect_v86:
   smsw    ax
   and     eax,1           ;CR0.PE bit
   ret
```

VM mode detection is mainly useful when writing DOS extenders or other programs that could be started either in plain real mode or in virtual mode from a protected mode system. An 'ordinary' bootloader shouldn't worry about this since the BIOS will not set up VM86 to read the bootsector ;)

Usage

VM86 can be very useful if one needs access to BIOS functions while in Protected mode. It is in fact necessary in order to set up video mode, as many modern card/chipsets lack support for the VBE3 protected mode interface, so setting up a VM86 task that will perform the 'set video mode' call is the preferred method.

Kernels below 1MB

It has been suggested that you map your kernel to a 'high' logical address (e.g. 0xC0000000) to avoid VM86 tasks interfering with it. This is especially important with large kernels which leave no room for VM86 code below 1MB, or when larger programs are expected to run within the VM86 box.

If all that is needed is a BIOS interrupt wrapper, then the following should work:

1. ensure that your 16bits code is on a separate page from any 32 bits code
2. enable paging
3. make kernel pages unwritable (and unreadable ?) for DPL3 and allow user-access only to those pages that contains your 16 bits code and pages that contains BIOS code or data.

Using VM86 for disk access

Though theoretically possible, it is probably not a good idea. Most BIOS disk access will include IRQ handlers, DMA transfers (uncontrollable from within your VM monitor), and may stick in VM86 task while the BIOS waits for an interrupt response while a 'good' driver would have let the CPU free for other processes.

Windows 9x suffered from system freezing during disk access. Often due to an INT13-through-VM86 problem.

See Also

Articles

- Virtual Monitor

Threads

- Creating vm86 task
- VM86 and INT10h
- kernel location & VM86

External Links

- Intel 80386 Reference Programmer's Manual Chapter 15 (http://www.cs.ucla.edu/~kohler/class/04f-aos/ref/i386/c15.htm)
- http://osdev.berlios.de/v86.html - by Tim Robinson
- A virtual-8086 mode monitor (http://my.execpc.com/~geezer/osd/pmode/v86mm.zip) - by Chris Giese
- x86emu (http://126.sytes.net/tmp/x86emu.zip) - a BSD style licensed vitual-8086 mode *emulator* -

very different from a *monitor*.
- [1] (http://gitorious.org/x86emu/) - x86emu and several other projects. See mdt for code getting VBE modes.
- Protected Mode BIOS Call Functionailty v2.0 (http://www.rohitab.com/discuss/topic/35103-switch-between-real-mode-and-protected-mode/) - by Napalm

Retrieved from "http://wiki.osdev.org/index.php?title=Virtual_8086_Mode&oldid=14629"
Categories: X86 CPU | Operating Modes
