Getting MS-DOS games to run on Windows 95: The interrupt flag

devblogs.microsoft.com/oldnewthing/20160411-00

April 11, 2016

Raymond Chen

In the flags register of the 80386 processor is a flag called the interrupt flag. If the flag is set, then the CPU will respond to hardware interrupts. If the flag is clear, then the CPU will ignore hardware interrupts. It was common for a game that used an MS-DOS extender to disable interrupts temporarily by doing this:

```
pushf   ; save flags (including whether interrupts are enabled)
cli     ; disable interrupts
..do something..
popf    ; restore original flags
        ; (this re-enables interrupts if interrupts were previously enabled)
```

There were other variations of this pattern, but most of them boiled down to the same basic idea.

One of the quirks of the 80386 architecture is that if the application does not have I/O privilege, the `popf` instruction does not restore the interrupt flag. Furthermore, the instruction doesn't trap, so the operating system doesn't know that the application tried and failed to change the interrupt flag. It just silently fails. This means that the above code fragment behaves differently depending on whether the application has I/O privilege. If it has I/O privilege, then the `popf` will restore interrupts to their previous state. But if it doesn't, then the `popf` instruction has no effect on the interrupt flag, and interrupts remain disabled.

MS-DOS extenders typically granted the client I/O privilege because they were designed to run in a single-tasking environment, so the client was allowed full control of the system. Windows, on the other hand, did not grant the client I/O privilege because it had to worry about multitasking. This means that in Windows, programs that used the above technique would hang because they disabled interrupts and never re-enabled them.

The DPMI specification specifically calls out this coding pattern as problematic and provides special services for managing the interrupt flag so that a client application can manipulate its interrupt flag in a way that the DPMI server understands. In practice, however, client applications were written on the assumption that they were running under the MS-DOS extender that they were packaged with, and they took shortcuts and just used the `popf` instruction because they knew that it would work. It never occurred to them that their preferred DPMI extender would not actually be the one in charge.

Except, of course, that it didn't work when the DPMI server was Windows.

One of my colleagues came up with a clever solution that addressed many cases of this problem. Since the `cli` instruction is privileged, it will trap. The trap handler for the `cli` instruction inspects the code preceding the `cli` instruction to see if it matches the pattern above or one of the common variations. If so, and the value on the top of the stack matches the virtual machine's current flags, then it assumes that the instruction that most recently executed was in fact the `pushf`. In that case, it copies the pushed interrupt flag to the pushed *trap* flag. In other words, if interrupts were previously enabled, then set the trap flag in the pushed flags.

In the 80386 architecture, the trap flag causes the processor to raise a trap exception after one instruction is executed. Its intended purpose is to allow debuggers to single-step through assembly code. The trick is to repurpose the flag: When the `popf` occurrs, the processor doesn't pop the interrupt flag, but it does pop the trap flag. After one instruction, the trap interrupt fires, and the kernel regains control. It recognizes that this trap interrupt is being used to regain control when somebody does a `popf` instruction in a failed attempt to re-enable interrupts. The kernel would then re-enable interrupts.

Result: As far as the application is concerned, the `popf` instruction successfully restored the interrupt flag despite all the warnings that said it wouldn't work. It was restored one instruction late, but it did eventually get restored.

That one weird trick rescued a lot of games from the "Doesn't work" category.

Related reading: What did Windows 3.1 do when you hit Ctrl+Alt+Del?, another case where the operating system used the trap flag.

Raymond Chen

Follow