CS490 Windows Internals Lab

Sept 27, 2013

1. Interrupts in Windows

Viewing IRQL in Kernel Debugger

If you are running the kernel debugger on Windows Server 2003, you can view a processor's IRQL with the !irql debugger command:

kd> !irql

```
0: kd> !irql
Debugger saved IRQL for processor 0x0 -- 0 (LOW_LEVEL)
0: kd> _
```

Note that there is a field called IRQL in a data structure called the processor control region (PCR) and its extension the processor control block (PRCB), which contain information about the state of each processor in the system. Portions of the PCR and PRCB structures are defined publicly in the Windows Device Driver Kit (DDK) header file Ntddk.h, so examine that file if you want a complete definition of these structures. You can view the contents of the PCR with the kernel debugger by using the !pcr command:

kd> !pcr

```
0: kd> !pcr
KPCR for Processor 0 at fffff80004802d00:
    Major 1 Minor 1
        NtTib.ExceptionList: fffff80005d50000
            NtTib.StackBase: fffff80005d51080
           NtTib.StackLimit: 000000003cdf0f8
         NtTib.SubSystemTib: fffff80004802d00
              NtTib.Version: 0000000004802e80
          NtTib.UserPointer: fffff800048034f0
              NtTib.SelfTib: 000000007ef55000

                    SelfPcr: 0000000000000000
                       Prcb: fffff80004802e80
                       Irql: 0000000000000000
                        IRR: 0000000000000000
                        IDR: 0000000000000000
              InterruptMode: 0000000000000000
                        IDT: 0000000000000000
                        GDT: 0000000000000000
                        TSS: 0000000000000000

              CurrentThread: fffff80004810c40
                 NextThread: 0000000000000000
                 IdleThread: fffff80004810c40

                  DpcQueue:
0: kd> _
```

Unfortunately, Windows does not maintain the Irql field on systems that do not use lazy IRQL, so, on most systems the field will always be 0.

Viewing IRQL/IRQ Assignments

You can view the contents of the IDT, including information on what trap handlers Windows has assigned to interrupts (including exceptions and IRQs), using the !idt kernel debugger command. The !idt command with no flags shows vectors that map to addresses in modules other than Ntoskrnl.exe. The following example shows what the output of the !idt command looks like:

kd> !idt

```
Dumping IDT:

00:     fffff80004685480 nt!KiDivideErrorFault
01:     fffff80004685580 nt!KiDebugTrapOrFault
02:     fffff80004685740 nt!KiNmiInterruptStart Stack = 0xFFFFF80005D62000

03:     fffff80004685ac0 nt!KiBreakpointTrap
04:     fffff80004685bc0 nt!KiOverflowTrap
05:     fffff80004685cc0 nt!KiBoundFault
06:     fffff80004685dc0 nt!KiInvalidOpcodeFault
07:     fffff80004686000 nt!KiNpxNotAvailableFault
08:     fffff800046860c0 nt!KiDoubleFaultAbort  Stack = 0xFFFFF80005D60000

09:     fffff80004686180 nt!KiNpxSegmentOverrunAbort
0a:     fffff80004686240 nt!KiInvalidTssFault
0b:     fffff80004686300 nt!KiSegmentNotPresentFault
0c:     fffff80004686440 nt!KiStackFault
0d:     fffff80004686580 nt!KiGeneralProtectionFault
0e:     fffff800046866c0 nt!KiPageFault
10:     fffff80004686a80 nt!KiFloatingErrorFault
11:     fffff80004686c00 nt!KiAlignmentFault
12:     fffff80004686d00 nt!KiMcheckAbort       Stack = 0xFFFFF80005D64000

13:     fffff80004687080 nt!KiXmmException
1f:     fffff800046653b0 nt!KiApcInterrupt
2c:     fffff80004687240 nt!KiRaiseAssertion
2d:     fffff80004687340 nt!KiDebugServiceTrap
2f:     fffff800046d1f10 nt!KiDpcInterrupt
37:     fffff80004c23090 hal!PicSpuriousService37 (KINTERRUPT fffff80004c23000)
3f:     fffff80004c23130 hal!PicSpuriousService37 (KINTERRUPT fffff80004c230a0)
51:     fffffa8004b12b10 fffffa80049bd5a0 (KINTERRUPT fffffa8004b12a80)
                         fffffa80049bd5a0 (KINTERRUPT fffffa8004b129c0)
                         fffffa80049bd5a0 (KINTERRUPT fffffa8004b12900)
                         fffffa8004ec05a0 (KINTERRUPT fffffa8004b12600)
                         fffffa8004ec05a0 (KINTERRUPT fffffa8004b12180)
52:     fffffa8004b12450 ndis!ndisMiniportMessageIsr (KINTERRUPT fffffa8004b123c
0)
60:     fffffa80061dad50 dxgkrnl!DpiFdoMessageInterruptRoutine (KINTERRUPT fffff
a80061dacc0)
62:     fffffa8004b12510 HDAudBus!HdaController::Isr (KINTERRUPT fffffa8004b1248
0)
70:     fffffa8004b12bd0 pci!ExpressRootPortMessageRoutine (KINTERRUPT fffffa800
4b12b40)
71:     fffffa80061daed0 i8042prt!I8042MouseInterruptService (KINTERRUPT fffffa8
0061dae40)
```

The left number is the interrupt number. You can see in the system, the mouse interrupt number in at 0x71.

2. Examining Interrupt Internals

Using the Kernel debugger, you can view details of an interrupt object, including its IRQL, ISR address, and custom interrupt dispatching code. For example, to see the details of the interrupt object of mouse interrupt in the above lab, try this command:

kd> dt nt!_KINTERRUPT fffffa80061daed0

```
0: kd> dt nt!_KINTERRUPT fffffa80061daed0
   +0x000 Type             : 0n21840
   +0x002 Size             : 0n-29368
   +0x008 InterruptListEntry : _LIST_ENTRY [ 0xcccccccc`5065ffff - 0xfffff800`04
802e80 ]
   +0x018 ServiceRoutine   : (null)
   +0x020 MessageServiceRoutine : (null)
   +0x028 MessageIndex     : 0
   +0x030 ServiceContext   : 0x00000000`00a00016 Void
   +0x038 SpinLock         : 0
   +0x040 TickCount        : 0
   +0x048 ActualLock       : 0xfffff880`063aca04  -> 0x53105089`48c48b48
   +0x050 DispatchAddress  : (null)
   +0x058 Vector           : 0
   +0x05c Irql             : 0 ''
   +0x05d SynchronizeIrql  : 0 ''
   +0x05e FloatingSave     : 0 ''
   +0x05f Connected        : 0 ''
   +0x060 Number           : 0x4fbf9f0
   +0x064 ShareVector      : 0x80 ''
   +0x065 Pad              : [3]  "???"
   +0x068 Mode             : 0 ( LevelSensitive )
   +0x06c Polarity         : 0 ( InterruptPolarityUnknown )
   +0x070 ServiceCount     : 0
   +0x074 DispatchCount    : 0
   +0x078 Rsvd1            : 0xfffffa80`04fbfb50
   +0x080 TrapFrame        : 0xfffff800`046843d0 _KTRAP_FRAME
   +0x088 Reserved         : 0x01000808`00000081 Void
   +0x090 DispatchCode     : [4] 0
0: kd>
```

To verify the IRQ, open Device Manager, locate the PS/2 mouse device, and view its resource assignments: