Breaking down the "Cl" in !irp

ntdebug 29 Oct 2012 4:18 PM | **2**

Hey there NTDEBUGGERS my name is Randy Monteleone and today we are going to talk about IRPs. In the past we have talked about the IRP structure in passing and showed a field here and there that can be pulled out and used to find answers to stalled IO. I was recently working on a debugger extension and found something interesting in the IRP I was looking at. I had been looking at a !irp output much like the one pictured below. I found that I was asking myself what do the "Success Error Cancel" fields mean?

After doing some digging and working with a few of my co-workers we found the mystery to the meaning behind these words and why we see them in our output. Lets break this IRP down starting with the ">" marker that indicates the current stack frame. In the output below you see this marker is indicating that we are working on something in partmgr.

```
1: kd> !irp fffffa809a1f3af0
Irp is active with 9 stacks 5 is current (= 0xfffffa809a1f3ce0)
Mdl=fffffa814e9c4f40: No System Buffer: Thread fffffa80d05b67e0:  Irp stack trace.
     cmd  flg cl Device   File     Completion-Context
[  0, 0]   0  0 00000000 00000000 00000000-00000000

                Args: 00000000 00000000 00000000 00000000
[  0, 0]   0  0 00000000 00000000 00000000-00000000

                Args: 00000000 00000000 00000000 00000000
[  0, 0]   0  0 00000000 00000000 00000000-00000000

                Args: 00000000 00000000 00000000 00000000
[  0, 0]   0  0 00000000 00000000 00000000-00000000

                Args: 00000000 00000000 00000000 00000000
>[  4,34]  1c e0 fffffa80920c0060 00000000 fffff880010301b0-00000000 Success Error Cancel <-- What are you trying to tell me?
       \Driver\Disk           partmgr!PmReadWriteCompletion
               Args: 00001000 00000000 14625000 00000000
[  4, 0]  1c e0 fffffa80920c0b90 00000000 fffff88001063010-fffffa80b01eec00 Success Error Cancel
       \Driver\partmgr        volmgr!VmpReadWriteCompletionRoutine
               Args: 3993d69568d 00000000 14625000 00000000
[  4, 0]   c e0 fffffa80b01eeab0 00000000 fffff88001d59150-fffffa80d05b2180 Success Error Cancel
       \Driver\volmgr         volsnap!VspRefCountCompletionRoutine
               Args: 00001000 00000000 3993d69568a 00000000
[  4, 0]   c e1 fffffa80d05b2030 00000000 fffff88001845344-fffff8800d52bb38 Success Error Cancel pending
       \Driver\volsnap        Ntfs!NtfsMasterIrpSyncCompletionRoutine
                Args: 00001000 00000000 14615000 00000000
[  4, 0]   0  0 fffffa80c2829030 fffffa80d04f18a0 00000000-00000000
       \FileSystem\Ntfs
                Args: 00001000 00000000 00002000 00000000
```

In the example below we see the "Success Error Cancel" fields and in this case we can also see that the pending flag has been set. The pending field indicates that STATUS_PENDING was returned to the caller. This is used so that I/O completion can determine whether or not to fully complete the I/O operation requested by the packet. Drivers can do this by calling **IoMarkIrpPending**.

Now look at the "cl" column as it holds the key to unlocking what "Success Error Cancel pending" really means.

```
1: kd> !irp 0xfffffa80`920c2340
Irp is active with 3 stacks 3 is current (= 0xfffffa80920c24a0)   <--- _IO_STACK_LOCATION
Mdl=fffffa814e9c4f40: No System Buffer: Thread 00000000: Irp stack trace.
     cmd  flg cl Device   File     Completion-Context
[  0, 0]   0  0 00000000 00000000 00000000-00000000

                Args: 00000000 00000000 00000000 00000000
[  f, 0]  1c  0 fffffa809209a060 00000000 fffff880010061a0-fffffa80920bfcc0
       \Driver\mpio        mpio!MPIOPdoCompletion
                Args: fffffa80920c22b0 00000000 00000000 fffffa80920bfcc0
>[  f, 0] 1c  e1 fffffa809209a060 00000000 fffff88001a01a00-fffffa80920c2190 Success Error Cancel pending
       \Driver\mpio        CLASSPNP!TransferPktComplete
```

```
Args: fffffa80920c22b0 00000000 00000000 fffffa80920bfcc0
```

Focusing on the cl column we see that our active stack frame is working in MPIO, but what is it trying to tell us about its status and intent? Well to figure that out we need to break down the value "e1" we see listed in our cl or Control column. This field is being used as a flag where the high and low parts represent two different values. We get this value in !irp from an _IO_STACK_LOCATION Control member. This can be found by dumping the stack location displayed by !irp.

```
1: kd> dt_IO_STACK_LOCATION 0xfffffa80920c24a0
nt!_IO_STACK_LOCATION
  +0x000 MajorFunction    : 0xf ''
  +0x001 MinorFunction    : 0 ''
  +0x002 Flags            : 0x1c ''
  +0x003 Control          : 0xe1 ''  < -- Control flag
  +0x008 Parameters       : <unnamed-tag>
  +0x028 DeviceObject     : 0xfffffa80`9209a060 _DEVICE_OBJECT
  +0x030 FileObject       : (null)
  +0x038 CompletionRoutine : 0xfffff880`01a01a00     long  CLASSPNP!TransferPktComplete+0
  +0x040 Context          : 0xfffffa80`920c2190 Void
```

So now that we know where we get this value we still have to decode what it means and how this value results in what we see in our !IRP output. Let's split our number into two, our high and low parts. In our example above we had the value e1. Taking our value apart gives us an E and a 1. The first bit of our value will indicates the last returned value as our IRP was processed. In our case this is a 1, meaning that our IRP is Pending. Thus we see the word pending at the end of current stack frame in !irp. There are three possible values for this bit, 0, 1 or 2. 0 being nothing, 1 being pending and 2 being Error Returned.

Moving to our high bits we are left with the "e". Lets take this number and convert it to binary. We end up with 1110. This upper number indicates which invoke types were requested for the completion routine for the driver listed on that stack frame. If we look at MSDN we see that **IoSetCompletionRoutine** takes in three bool values to set this flag. These options specify whether the completion routine is called if the IRP is completed with that corresponding status.

```
VOID IoSetCompletionRoutine(
  _In_       PIRP Irp,
  _In_opt_   PIO_COMPLETION_ROUTINE CompletionRoutine,
  _In_opt_   PVOID Context,
  _In_       BOOLEAN InvokeOnSuccess,
  _In_       BOOLEAN InvokeOnError,
  _In_       BOOLEAN InvokeOnCancel
);
```

Doing some source review I was able to pin together how these values translate to the !irp output. If we look back at our binary value of "e" we see that we have a set of bits that get set based on the what the driver wanted to do when an IRP was completed with one of our defined status values.

```
Cancel   = 2
Success  = 4
Error    = 8
```

Add each of these values up and the sum is 14 or e in hex. Going back to our binary "1110" we see that our first bit is set to zero or off. This would be the pending and Error Returned values that I mentioned above. The next three bits represent the corresponding Cancel, Success and Error bool values passed to the driver at the time IoSetCompletionRoutine was called for this stack location.

Example : IoSetCompletionRoutine(pirp, pCompletionRoutine,pContex, True,False,True); would yield a value of 6.

Remember the important thing here is **not** that the !irp output is trying to tell us that one of these things happened. It's telling us that this driver would like to be notified if one of those things **does** happen. This area also provides us with information if pending or or an error is returned.

Well that's all I have for now, for more documentation for the _IRP and _IO_STACK_LOCATION structures please see the following links to MSDN.

More on IRP: **http://msdn.microsoft.com/en-us/library/windows/hardware/ff550694(v=vs.85).aspx**
More on _IO_STACK_LOCATION: **http://msdn.microsoft.com/en-us/library/windows/hardware/ff550659(v=vs.85).aspx**

Comments

Miroslav Reisinger
30 Oct 2012 3:07 PM

I always wondered about the fields you just described Randy, thanks!

sergmat
1 Nov 2012 5:43 AM

Add post to "I/O Request Packet IRP (en-US)" TechNet wiki article.

social.technet.microsoft.com/.../6697.io-request-packet-irp-en-us.aspx

[Thanks!]