You must flush GDI operations when switching between direct access and GDI access, and direct access includes other parts of GDI

devblogs.microsoft.com/oldnewthing/20100923-00

September 23, 2010

Raymond Chen

A customer was running into problems when accessing the pixels of a DIB section. They used the `HANDLE` parameter to `CreateDIBSection` and created two bitmaps from the same underlying memory. Those two bitmaps were then selected into corresponding DCs, and the customer found that changes to the pixels performed by writing via one DC were not visible when read from the other DC.

The customer pointed out this clause in MSDN:

> You need to guarantee that the GDI subsystem has completed any drawing to a bitmap created by `CreateDIBSection` before you draw to the bitmap yourself. Access to the bitmap must be synchronized. Do this by calling the `GdiFlush` function. This applies to any use of the pointer to the bitmap bit values, including passing the pointer in calls to functions such as `SetDIBits`.

The customer said, "The description says that it applies to cases where you modify the bits yourself through the direct memory pointer. But all of our access is performed through HDCs; I would think GDI is smart enough to handle that, but we've found that we still need to call `GdiFlush` to get the two DCs back in sync."

What you ask GDI to do you have done yourself. That's why the documentation say *any use of the pointer*. Sort of like in law, where in many causes you can be punished for "doing X or causing X to be done." If you induce somebody else to do X, you're in violation as much as if you had done X yourself.

I doubt that every call to GDI ends with a big loop that checks whether the bits it just modified also belong to some other GDI bitmap in the system.

```
GDIFinishAPI(HDC hdc)
{
 if (IsDIBSection(GetCurrentObject(hdc, OBJ_BITMAP), &ds)) {
  EnumGdiObjects(FlushIfOverlap, &ds));
 }
}
FlushIfOverlap(HGDIOBJ h, DIBSECTION *pds)
{
 if (IsDIBSection(h, &ds) &&
     DIBSectionsReferToSameUnderlyingBits(pds, &ds)) {
  GdiFlush();
 }
}
```

That would seriously slow down all DIB section operations to cover a rare scenario that most people don't realize is even possible to create. Not the best engineering tradeoff.

The point of the documentation is is that if you ask GDI to mess with the bits in the bitmap via the `HDC`, you must call `GdiFlush` before anybody else tries to access those bits, even if that "somebody else" is another part of GDI. The example of `SetDIBits` is an attempt to capture the sense of this requirement.

Translating into this specific scenario: You must flush the pending changes whenever you switch between "GDI accesses bits through the DIB section created by this handle" and "the bits are accessed by anybody else." And "anybody else" could be "GDI accesses bits through the DIB section created by a different handle."

Bonus chatter: What's the deal with `GdiFlush` anyway?

As a performance optimization, GDI performs "batching" of operations. When you ask GDI to perform an operation, it doesn't always do it right away. Instead, it may choose to store the action in a buffer, and when the buffer gets full, it "flushes the batch" and sends the commands that it had been saving up into kernel mode for execution. (This idea of buffering up operations and submitting them as a batch is hardly new to GDI. The C stdio library does it, and in networking, a variation of it goes by the name Nagle's Algorithm.)

GDI also flushes the batch when necessary in order to preserve semantics; for example, if you call `GradientFill` and follow it with a call to `GetPixel`, GDI needs to flush out the `GradientFill` before issuing the `GetPixel` so that the pixels that get read match the pixels that were written. (A much more common case of just-in-time flushing is where you `BitBlt` the results out of the bitmap into another device context.)

This behind-the-scenes optimization works great with one exception: DIB sections. Since the memory for DIB sections is directly visible, GDI doesn't get a chance to sneak a call to `GdiFlush` before you issue your "mov eax, [esi]" instruction. Hence the clause in MSDN

explaining that when you switch between GDI access and direct access, you need to call `GdiFlush` to get all pending operations out of the buffer so that the bits in memory match the operations you performed.

Many years ago, we saw another case where we had to compensate for GDI batching.

Raymond Chen

Follow