Using the powers of mathematics to simplify multi-level comparisons

devblogs.microsoft.com/oldnewthing/20050413-56

April 13, 2005

Raymond Chen

What a boring title.

Often you'll find yourself needing to perform a multi-level comparison. The most common example of this is performing a version check when there are major and minor version numbers involved. Bad version number checks are one of the most common sources of errors.

If you're comparing version numbers, you can use the `VerifyVersionInfo` function to do the version check for you, assuming you don't need to run on operating systems prior to Windows 2000.

Instead of writing a multi-level comparison, you can pack the values into a single comparison. Consider:

```
inline unsigned __int64
MakeUINT64(DWORD Low, DWORD High)
{
  ULARGE_INTEGER Value;
  Value.LowPart = Low;
  Value.HighPart = High;
  return Value.QuadPart;
}
BOOL IsVersionAtLeast(DWORD Major, DWORD Minor,
                      DWORD MajorDesired, DWORD MinorDesired)
{
  return MakeUINT64(Minor, Major) >= MakeUINT64(MinorDesired, MajorDesired);
}
```

What happened here?

We took the two 32-bit values and combined them into a large 64-bit value, putting the most significant portion in the high-order part and the less significant portion in the lower-order part.

Then we sit back and let the power of mathematics do our work for us. If you remember the rules for comparisons from grade school, you'll realize that they exactly match the rules we want to apply to our multi-level comparison. Compare the major values; if different, then that's the result. Otherwise, compare the minor values.

If you still don't believe it, look at the generated code:

```
  00000 8b 44 24 04      mov     eax, DWORD PTR _Major$[esp-4]
  00004 3b 44 24 0c      cmp     eax, DWORD PTR _MajorDesired$[esp-4]
  00008 8b 4c 24 08      mov     ecx, DWORD PTR _Minor$[esp-4]
  0000c 8b 54 24 10      mov     edx, DWORD PTR _MinorDesired$[esp-4]
  00010 72 0b            jb      SHORT $L48307
  00012 77 04            ja      SHORT $L48317
  00014 3b ca            cmp     ecx, edx
  00016 72 05            jb      SHORT $L48307
$L48317:
  00018 33 c0            xor     eax, eax
  0001a 40               inc     eax
  0001b eb 02            jmp     SHORT $L48308
$L48307:
  0001d 33 c0            xor     eax, eax
$L48308:
  0001f c2 10 00         ret     16                    ; 00000010H
```

The code generated by the compiler is equivalent to

```
BOOL IsVersionAtLeastEquiv(DWORD Major, DWORD Minor,
                      DWORD MajorDesired, DWORD MinorDesired)
{
 if (Major < MajorDesired) return FALSE;
 if (Major > MajorDesired) return TRUE;
 if (Minor < MinorDesired) return FALSE;
 return TRUE;
}
```

In fact, if you had written the code the (error-prone) old-fashioned way, you would have gotten this:

```
BOOL IsVersionAtLeast2(DWORD Major, DWORD Minor,
                       DWORD MajorDesired, DWORD MinorDesired)
{
  return Major > MajorDesired ||
   (Major == MajorDesired && Minor >= MinorDesired);
}
  00000 55              push    ebp
  00001 8b ec           mov     ebp, esp
  00003 8b 45 08        mov     eax, DWORD PTR _Major$[ebp]
  00006 3b 45 10        cmp     eax, DWORD PTR _MajorDesired$[ebp]
  00009 77 0e           ja      SHORT $L48329
  0000b 75 08           jne     SHORT $L48328
  0000d 8b 45 0c        mov     eax, DWORD PTR _Minor$[ebp]
  00010 3b 45 14        cmp     eax, DWORD PTR _MinorDesired$[ebp]
  00013 73 04           jae     SHORT $L48329
$L48328:
  00015 33 c0           xor     eax, eax
  00017 eb 03           jmp     SHORT $L48330
$L48329:
  00019 33 c0           xor     eax, eax
  0001b 40              inc     eax
$L48330:
  0001c 5d              pop     ebp
  0001d c2 10 00        ret     16                    ; 00000010H
```

which is, as you can see, functionally identical to both previous versions.

You can also pack the values into smaller units, provided you know that there will be no overflow or truncation. For example, if you know that the Major and Minor values will never exceed 65535, you could have used the following:

```
BOOL SmallIsVersionAtLeast(WORD Major, WORD Minor,
                           WORD MajorDesired, WORD MinorDesired)
{
 return MAKELONG(Minor, Major) >= MAKELONG(MinorDesired, MajorDesired);
}
 00000 0f b7 44 24 0c   movzx   eax, WORD PTR _MajorDesired$[esp-4]
 00005 0f b7 4c 24 10   movzx   ecx, WORD PTR _MinorDesired$[esp-4]
 0000a 0f b7 54 24 08   movzx   edx, WORD PTR _Minor$[esp-4]
 0000f c1 e0 10         shl     eax, 16                       ; 00000010H
 00012 0b c1            or      eax, ecx
 00014 0f b7 4c 24 04   movzx   ecx, WORD PTR _Major$[esp-4]
 00019 c1 e1 10         shl     ecx, 16                       ; 00000010H
 0001c 0b ca            or      ecx, edx
 0001e 33 d2            xor     edx, edx
 00020 3b c8            cmp     ecx, eax
 00022 0f 9d c2         setge   dl
 00025 8b c2            mov     eax, edx
 00027 c2 10 00         ret     16                    ; 00000010H
```

And if you know that the versions will never exceed 255, then you can go even smaller:

```
BOOL TinyIsVersionAtLeast(BYTE Major, BYTE Minor,
                          BYTE MajorDesired, BYTE MinorDesired)
{
 return MAKEWORD(Minor, Major) >= MAKEWORD(MinorDesired, MajorDesired);
}
  00000 33 c0            xor     eax, eax
  00002 8a 64 24 0c      mov     ah, BYTE PTR _MajorDesired$[esp-4]
  00006 33 c9            xor     ecx, ecx
  00008 8a 6c 24 04      mov     ch, BYTE PTR _Major$[esp-4]
  0000c 8a 44 24 10      mov     al, BYTE PTR _MinorDesired$[esp-4]
  00010 8a 4c 24 08      mov     cl, BYTE PTR _Minor$[esp-4]
  00014 66 3b c8         cmp     cx, ax
  00017 1b c0            sbb     eax, eax
  00019 40               inc     eax
  0001a c2 10 00         ret     16                    ; 00000010H
```

Why would you ever need to go smaller if the original version works anyway? Because you might want to make a three-way or four-way comparison, and packing the values smaller allows you to squeeze more keys into the comparison.

```
BOOL IsVersionBuildAtLeast(
    WORD Major, WORD Minor, DWORD Build,
    WORD MajorDesired, WORD MinorDesired, DWORD BuildDesired)
{
 return MakeUINT64(Build, MAKELONG(Minor, Major)) >=
  MakeUINT64(Build, MAKELONG(MinorDesired, MajorDesired));
}
```

By packing the major version, minor version, and build number into a single 64-bit value, a single comparison operation will compare all three at once. Compare this to the complicated (and teetering-towards unreadable) chain of comparisons you would normally have to write:

```
  return Major > MajorDesired ||
   (Major == MajorDesired &&
    (Minor >= MinorDesired ||
     (Minor == MinorDesired && Build >= BuildDesired)));
```

Raymond Chen

Follow