QueryPerformanceCounter is not a source for unique identifiers

devblogs.microsoft.com/oldnewthing/20070705-00

July 5, 2007

Raymond Chen

This article happened to catch my eye:

> I needed to generate some unique number in my application. I could use GUID, but it was too large for me (I need to keep lots of unique identifiers). I found something like this:
>
> ```
> [System.Runtime.InteropServices.DllImport("Kernel32.dll")]
> private static extern int
>     QueryPerformanceFrequency(ref System.Int64 frequency);
> [System.Runtime.InteropServices.DllImport("Kernel32.dll")]
> private static extern int
>     QueryPerformanceCounter(ref System.Int64 performanceCount);
> public static long GenerateUniqueId()
> {
>       System.Int64 id = 0;
>       QueryPerformanceFrequency(ref id);
>       QueryPerformanceCounter(ref id);
>       return id;
> }
> ```
>
> This code generates Int64 (long) unique number (at least I hope it is unique). The uniqueness is in the scope of process. So two processes can generate the same number, but it should be always unique in a single process (I am not sure about two threads invoking the same GenerateUniqueId() method.

`QueryPerformanceCounter` retrieves the current value of the high-resolution performance counter, but there is no guarantee that every call to the function will return a different number.

The frequency of the high-resolution performance counter is determined by the HAL. You might think that the `RDTSC` instruction would be perfect for this purpose, since it returns the number of CPU clock ticks, a value that always increases at a very high rate. But there are many problems with `RDTSC`. For example, variable-speed processors mean that the rate at

which CPU clock elapse varies over time. A million clock ticks might take one millisecond when the computer is running on wall power, but two milliseconds when running on battery power.

If the HAL can't use `RDTSC`, what does it use instead? Well, as I said, it's up to the HAL to find something suitable. Older motherboards have to make do with the programmable interval timer which runs at 1,193,182 ticks per second (approximately 0.8 microseconds per tick). Newer motherboards can use the ACPI timer which runs at 3,579,545 ticks per second (approximately 0.3 microseconds per tick).

One of the machines in my office uses the ACPI timer for its high-resolution performance counter, so I threw together a quick program to see how close I can get to outracing the ACPI timer by calling `QueryPerformanceCounter` in rapid succession. With a 1.80GHz processor, the computer manages to call `QueryPerformanceCounter` quickly enough that only four ticks of the ACPI timer elapse between consecutive calls. We're getting into shouting range of being able to call `QueryPerformanceCounter` twice and getting the same value back from the ACPI timer. Of course, if the computer had been using the programmable interval timer, it would have been within spitting distance, and upgrading to a 3GHz processor would have taken us over the top.

In other words, you may be lucky today that your CPU isn't fast enough to call `QueryPerformanceCounter` twice and get the same value back, but it sure looks like we're threatening to get there soon.

Then again, all this back-of-the-envelope calculation is superfluous. All you need is a machine with multiple processors. Get two of the processors to call `QueryPerformanceCounter` at the same time (or nearly so), and they'll get the same timer value back.

If you want to generate unique 64-bit values, you can just use `InterlockedIncrement64`.

Raymond Chen

Follow