MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
OCT 2, 2017 3:00 AM PDT

MASTER C#

When to use the volatile keyword in C#

How to take advantage of the volatile keyword in C# to ensure that concurrent threads get the latest value of an object

The optimization techniques used by the JIT (just-in-time) compiler in the Common Language Runtime might lead to unpredictable results when your .Net program is trying to perform non-volatile reads of data in a multithreaded scenario. In this article we'll look at the differences between volatile and non-volatile memory access, the role of the volatile keyword in C#, and how the volatile keyword should be used.

I will provide some code examples in C# to illustrate the concepts. To understand how the volatile keyword works, first we need to understand how the JIT compiler optimization strategy works in .Net.

[Discover 2017's best open source software for enterprise: The Bossie Award winners. | Track the latest trends in open source with InfoWorld's Open Source Report newsletter.]

Understanding JIT compiler optimizations

It should be noted that the JIT compiler will, as part of an optimization strategy, change the order of the reads and writes in a way that does not change the meaning and eventual output of the program. This is illustrated in the code snippet given below.


```
x = 0;
x = 1;
```

The above snippet of code can be changed to the following—while preserving the program's original semantics.

```
x = 1;
```

The JIT compiler can also apply a concept called "constant propagation" to optimize the following code.

```
x = 1;
y = x;
```

The above code snippet can be changed to the following—again while preserving the original semantics of the program.

```
x = 1;
y = 1;
```

Volatile vs. non-volatile memory access

The memory model of modern-day systems is quite complicated. You have processor registers, various levels of caches, and main memory shared by multiple processors. When your program executes, the processor may cache the data and then access this data from the cache when it is requested by the executing thread. Updates and reads of this data might run against the cached version of the data, while the main memory is updated at a later point in time. This model of memory usage has consequences for multithreaded applications.

When one thread is interacting with the data in the cache, and a second thread tries to read the same data concurrently, the second thread might read an outdated version of the data from the main memory. This is because when the value of a non-volatile object is updated, the change is made in the cache of the executing thread and not in the main memory. However, when the value of a volatile object is updated, not only is the change made in the cache of the executing thread, but this cache is then flushed to the main memory. And when the value of a volatile object is read, the thread refreshes its cache and and reads the updated value.

Using the volatile keyword in C#

The volatile keyword in C# is used to inform the JIT compiler that the value of the variable should never be cached because it might be changed by the operating system, the hardware, or a concurrently executing thread. The compiler thus avoids using any optimizations on the variable that might lead to data conflicts, i.e. to different threads accessing different values of the variable.

When you mark an object or a variable as volatile, it becomes a candidate for volatile reads and writes. It should be noted that in C# all memory writes are volatile irrespective of whether you are writing data to a volatile or a non-volatile object. However, the ambiguity happens when you are reading data. When you are reading data that is non-volatile, the executing thread may or may not always get the latest value. If the object is volatile, the thread always gets the most up-to-date value.

You can declare a variable as volatile by preceding it with the `volatile` keyword. The following code snippet illustrates this.

```
class Program
    {
        public volatile int i;
        static void Main(string[] args)
        {
            //Write your code here
        }
    }
```

You can use the `volatile` keyword with any reference, pointer, and enum types. You can also use the volatile modifier with byte, short, int, char, float, and bool types. It should be noted that local variables cannot be declared as volatile. When you specify a reference type object as volatile, only the pointer (a 32-bit integer that points to the location in memory where the object is actually stored) is volatile, not the value of the instance. Also, a double variable cannot be volatile because it is 64 bits in size, larger than the word size on x86 systems. If you need to make a double variable volatile, you should wrap it inside in class. You can do this easily by creating a wrapper class as shown in the code snippet below.


```
public class VolatileDoubleDemo
{
    private volatile WrappedVolatileDouble volatileData;
}
public class WrappedVolatileDouble
{
    public double Data { get; set; }
```

However, note the limitation of the above code example. Although you would have the latest value of the `volatileData` reference pointer, you are not guaranteed the latest value of the `Data` property. The work around for this is to make the `WrappedVolatileDouble` type immutable.

Although the volatile keyword can help you in thread safety in certain situations, it is not a solution to all of your thread concurrency issues. You should know that marking a variable or an object as volatile does not mean you don't need to use the lock keyword. The volatile keyword is not a substitute for the lock keyword. It is only there to help you avoid data conflicts when you have multiple threads trying to access the same data.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

How to choose a low-code development platform

YOU MAY ALSO LIKE

This is your last article this month. Register now for free access.

Tim O'Reilly: the golden age of the programmer is over

3 creative ways to obtain cloud talent

What to expect in Java 18

Register Now

What's this?

What is cloud-native? The modern way to develop software

No one wants to manage Kubernetes anymore

What Windows 11 means for developers

Meta or micro cloud architecture? You need

Tips for agile and devops teams in a hybrid work

12 ways to make really bad technology decisions

Jetpack Compose for Android turns GA

GitHub Copilot is 'unacceptable and unjust,' says Free Software Foundation

Join the most important gathering of CIOs & IT executives at Gartner IT Symposium/Xpo™ 2021.

Getting a Grip on Basic Cyber Hygiene with the CIS Controls

Preparing Your Technology Foundation for a New Hybrid World

NETSCOUT Visibility Without Borders keeps you one step ahead by helping you to quickly mitigate cyberattacks and resolve network performance issues.

Drive recurring revenue streams, build customer intimacy, and supercharge your service business with our remote monitoring & management platform, EcoStruxure IT.

IT solution providers give their customers up to 50% lower total cost of ownership with Smart-UPS Lithium-ion and get more revenue in the process.

Your cloud, your way: Why Cloud Verified matters