Thomas Drehman

Developing Concurrent Software

Why Atomics Solve Double-Checked Locking Problem in C++11

It has been about 14 years since The “Double-Checked Locking is Broken” Declaration was signed by over a dozen computer experts in both academia and industry. While their collective expertise was in Java, the declaration also made it clear that double-checked locking was broken in C++ as well. Since this declaration, the 2011 changes to the C++ standard enabled double check locking.

Double-checked Locking is an important tool in concurrent programming as it is an “efficient way to implement lazy initialization in a multithreaded environment”.[[1]](#footnote-1) Consider a multithreaded compatible class that has a function that creates a new “helper” object which would run code. The code would create a helper if there was no available helper. The code would look something like:

class Example{

private helper == “NULL”; // Class creates no helpers when initiated

public synchronized helper getHelper(){

if(helper == “NULL”)

helper = new helper;

return helper;}

// Other class functions and data

}

Note that the synchronous function allows multiple threads to call the getHelper function. On the surface this looks like it will work.

Unfortunately, the code has some major logical problems. Since compilers can reorder code to make it more parallel and the fact that either having a helper or not having a helper are both valid states, the getHelper function can create a new helper improperly or fail to create one properly. This is because another thread called the helper constructor within the if statement or another thread called the helper destructor but had not finished destructing the helper object when the first thread had reached the if statement, respectively.

The 2011 changes to the C++ standard fix this problem with the implementation of atomics. With atomics, some execution order can be established which will prevent the out of order execution behaviors that made double-checked locking broken previously. With the test class Example below, synchronizes with behavior exists between any threads that use the function Example::getHelper(). Notice the change was to make helper an atomic variable.

class Example{

private atomic helper == “NULL”;

public synchronized helper getHelper(){

if(helper == “NULL”)

helper = new helper;

return helper;}

}

Now when two threads both try to call getHelper() simultaneously, one thread is locked out of access to helper until the other thread has finished using it. This serializes the code slightly, but secures the consistency of helper.

Although double-checked locking was broken back in the early 2000s when the declaration that double-checked locking is broken was made, C++ has changed to fix this and other problems. With atomics in the C++ standard library, the declaration no longer applies to C++.

1. The “Double-Checked Locking is Broken” Declaration, Bacon et al. [↑](#footnote-ref-1)