

Re: WaitForSingleObject() will not deadlock

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2007-07/msg00315.html>

- *From:* "Doug Harrison [MVP]" <dsh@xxxxxxxx>
 - *Date:* Tue, 03 Jul 2007 13:41:53 -0500
-

On Tue, 03 Jul 2007 13:53:09 -0400, Joseph M. Newcomer <newcomer@xxxxxxxxxxxx> wrote:

Strictly speaking, shared variables should be declared 'volatile' to ensure the compiler has not kept a copy around, and this is completely orthogonal to the concept of locking; it appears that the Microsoft C/C++ compiler is very conservative about code motions across functions and therefore seems less subject to this kind of error, although the formal requirements of the C/C++ language would dictate that the use of 'volatile' is mandatory to ensure the compiler has not done something bad.

The C and C++ standards don't address the subject of multithreading, so it doesn't make sense to use them to reason about what compilers require to deal with multithreading.

(Most optimizing compilers I worked on over the years in fact would have required 'volatile', had that been part of the languages we were compiling; instead, we had to add "optimization fences", which were different from what are now known as "memory fences", to break the optimizer, so our lock calls were macros that both acquired the lock and broke the optimizations by injecting an optimization fence into the intermediate instruction stream inside the compiler).

Any compiler used for multithreaded programming that screws up the following is broken:

```
int x;

// Let f and g be called concurrently from different threads.

void f()
{
lock mutex;
use x;
unlock mutex;
// Must not cache x across mutex release/acquire
lock mutex;
```

Re: WaitForSingleObject() will not deadlock

```
use x;  
unlock mutex;  
}
```

```
void g()  
{  
lock mutex;  
++x;  
unlock mutex;  
}
```

Please don't use ancient compilers that could not be used for multithreaded programming to reason about compilers that can. (As I've posted several times before, getting the lock/unlock semantics right typically comes for free, since they exist in opaque system DLLs, that the optimizer cannot see into. Therefore, it has to assume x is reachable from those functions and that they can modify it.)

--

Doug Harrison
Visual C++ MVP

.