

## Re: Should I use mutex in this context?

---

*Source:* <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2008-10/msg00874.html>

---

- *From:* "Liviu" <lab2k1@xxxxxxxx>
  - *Date:* Fri, 31 Oct 2008 09:53:10 -0500
- 

"Tommy" <tommy767@xxxxxxxx> wrote in message  
<news:u5Wvs%23yOJHA.4328@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>

Liviu wrote:

It `_is_` true if X is `_not_` declared as "volatile" and if the compiler can ascertain that it's non-0 at the beginning of the loop. For example, given code like

```
int X;

void f()
{
  int n = 0;
  for(X = 1; X; )
    n++;
  printf("n = %d\n", n);
}
```

[snip]

This is all understood. But who will do that? It doesn't make sense to provide examples that are extreme and outside the scope of the stated design question.

Remember the stated OP question/problem set:

My program has two threads; one main thread and other working thread. My main thread asks the worker thread to terminate if system is shutting down by making true a boolean value which is a global shared resource. The main thread only writes to the global value and the worker thread only reads the same value. In this context, is mutex use necessary?

## Re: Should I use mutex in this context?

- A global shared boolean flag
- one writer (main) thread,
- one worker reader thread,
- All he cares for is for a graceful shutdown.

In this case, you don't need a MUTEX or any other synchronization kernel object, volatile or otherwise.

Sigh. Yes, you do.

We are talking about a simple example:

```
bool stupidflag = 0;
void stupid_thread()
{
while(!stupidflag)
{
... stupid_flag is NOT referenced or visible ...
```

Not visible?

```
}
}
```

Please point the distinguishing difference between your `stupid_thread()` and my `f()` above. Point, in both cases, is whether the compiler can ascertain what the value of your `stupid_flag` (or `X`) is when the function is entered. If it can, then the entire loop can be optimized to either empty or infinite – as long as `stupidflag` is `_not_` declared "volatile".

I have fail to see assembly in either the old and new Microsoft compilers for non-optimizing OR full-optimize assembly code where it fundamentally alters the logic so that it creates a BUG with effective code that emulates the following flawed code: [...]

The C world doesn't end with the MS compilers. MS' own compilers don't end at the current version. Just because something happens to work for you now, that alone doesn't make it right.

```
if (!stupidflag) do {
... stupid_flag is NOT referenced or visible ...
} while (1)
```

Re: Should I use mutex in this context?

## Re: Should I use mutex in this context?

What I am seeing instead in optimized compiled assembly code that keeps the end result the same – an transparent optimization that does not change the outcome using basically this flow:

```
if (!stupidflag) do {  
... stupid_flag is NOT referenced or visible ...  
} while (!stupidflag)
```

I'm sure we can change scenarios around to prove a point and even make the flawed `do { ... } while(1)` loop,

C is neither an experimental science, nor a matter of opinion. Please refer to the standard I referenced and explain under what interpretation that would be a "flaw". On the contrary, it would be a perfectly legal optimization per...

5.1.2.3.3. In the abstract machine, all expressions are evaluated as specified by the semantics. An actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no needed side effects are produced (including any caused by calling a function or accessing a volatile object).

All my point is that to answer this OP question by stating a MUTEX (or some other sync object) is required – well, its not quite true.

As it stands, what you imply is simply false. It may `_happen_` that the other code you posted elsewhere in a reply to Ben could work for you with a given compiler, by chance or because the respective compiler cannot yet see through some opaque function calls inside the loop, such as `printf` or `Sleep`. Yet, technically there is nothing to prevent the next version of the compiler from being smarter and realizing that neither call touches `PoorManSwitch`, and thus optimize the entire loop condition away.

It is your prerogative of course to write your own code relying on compiler specific quirks or undocumented hacks. But it is not sound advice to give to others, at least not without a clear indication of what your hidden assumptions and ensuing risks are.

Liviu

P.S. If you declared your `stupid_flag` "volatile" then most of this `stupid_thread` would have probably run its course already, instead of being stuck in a loop ;-)

Just curious, but where do you think that "volatile" `_is_` warranted?

Re: Should I use mutex in this context?