Subject: Re: Use same variable in different threads
Posted by gprentice on Sat, 11 Dec 2010 00:37:17 GMT
View Forum Message <> Reply to Message

Didier wrote on Sat, 11 December 2010 06:50Hi all,

Quote:
Actually, on second thoughts I think the code as written is fine without volatile because the mutex forces the shared data to be updated in memory. Possibly an atomic variable that is read or written outside of a mutex region is more likely to need volatile.

Yes this is the case, and is what I explained in my earlier post.

Just to be clear, apparently there are situations when volatile is useful
http://www.drdobbs.com/high-performance-computing/212701484

but in general, you need a memory barrier when accessing shared data.

After googling for a while, I haven't found any clear explanation of how a mutex solves the caching and memory visibility issues. I'm guessing that both acquire mutex and release mutex flush the entire memory cache for all CPUs/caches and prevent hardware re-ordering across the memory barrier - which deals with the hardware problem. For dealing with compiler/linker optimisation, I'm guessing that the call to the "external" acquire/release mutex function forces the compiler not to cache values across that call because it can't tell what memory that function call is going to access. Explanations of how a mutex works don't seem to mention these issues.

That AtomicVar class looks like a good idea. Should the copy constructor be
AtomicVar(const AtomicVar& p) { AtomicWrite(val, AtomicRead(p.val)); }

Graeme