

## Re: AfxBeginThread startup times and overhead

**Source:** <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2004-05/2765.html>

---

**From:** Joseph M. Newcomer (*newcomer\_at\_flounder.com*)

**Date:** 05/30/04

Date: Sun, 30 May 2004 19:48:50 -0400

See below...

On Sat, 29 May 2004 23:29:20 -0400, A. J. Rappaport  
<please.reply.in.newsgroup.only@spamsucks.com> wrote:

>On Sat, 29 May 2004 22:09:37 -0400, Joseph M. Newcomer

><newcomer@flounder.com> wrote:

>

>>>for ( i = 0; i < 3; ++i )

>>>{

>>> AfxBeginThread( ThreadStartingFct, &i )

>>> Sleep( 5 );

>>>}

>>>

>>>unsigned int ThreadStartingFct( LPVOID seq )

>>>{

>>> int localSeq = seq;

>>\*\*\*\*

>>This code won't even compile; I have no idea how you could use it. And according to what

>>you did, if it compiles, it tries to store into an int-sized variable a pointer-sized

>>value, which would lose data.

>

>Of course you are right, I meant to say

>

> int\* localSeq = (int\*)seq;

>

> ... which in fact is what's in my code.

\*\*\*\*

And which for the reasons I've already pointed out, is going to in general produce nonsense, and in the worst case crash in some horrible fashion (perhaps an access fault, but it could be just about anything). Reason: you passed the address of a local variable into a thread. This is, with some very rare and esoteric exceptions which involve incredible care, a mistake. The amount of effort required to make it valid is usually not worth the expenditure of time, and in most cases it ends up producing an incorrect, inefficient, or overly-complex solution to a simple problem. By the time the thread executes, the stack location is either invalid or gone entirely. And the Sleep(5) won't prevent this. It only gives you, in a unique situation, the illusion that anything worked at all. In the general situation, as I pointed out, it is merely syntactic noise.

\*\*\*\*

>

>As for spinning on a flag, I understand that's not the "right" way,  
>but I'm a little curious why you consider that such a bad design in  
>this specific simple example.

\*\*\*\*

Because the thread which is spinning on the flag will consume 100% of its timeslice each time it runs, consuming CPU time that could have been used to run the thread instead. Since the thread can always run, it always consumes as much of the CPU as it can get.

\*\*\*\*

>Your favored approach of mutexes,  
>semaphores, etc. are by their nature blocking mechanisms, so  
>presumably the same types of lockups can happen if the semaphore  
>doesn't show up or the mutex doesn't get released as if the flag  
>doesn't get cleared.

\*\*\*\*

Right. In either case, your program would be erroneous and would have to be fixed. The difference is that the thread would block, instead of putting your app into a state where it consumes 100% of the CPU, interfering with everything else that is going on. But in general it is always a mistake to introduce such blocking, since it negates the advantage of threading. A thread should be able to execute asynchronously, which means it must have no sequencing dependency once it has started. If there are sequencing dependencies, e.g., a UI thread which has to have its message pump started, you introduce other mechanisms to guarantee this has occurred, which are also necessarily timing-independent.

\*\*\*\*

>

>As for the rest of what you all have advised me, you have given me a  
>lot of help and a lot to think about and I thank you.

Joseph M. Newcomer [MVP]

email: [newcomer@flounder.com](mailto:newcomer@flounder.com)

Web: <http://www.flounder.com>

MVP Tips: [http://www.flounder.com/mvp\\_tips.htm](http://www.flounder.com/mvp_tips.htm)