

Re: AfxBeginThread startup times and overhead

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2004-05/2699.html>

From: Joseph M. Newcomer (*newcomer_at_flounder.com*)

Date: 05/30/04

Date: Sat, 29 May 2004 22:09:37 -0400

The presence of the Sleep(5) is pretty meaningless. It doesn't do anything worthwhile, and if it appears to that it makes your program work, you are mistaken. Perhaps on your system, with the load of just testing your app, you may have the illusion that something is happening, but on a real system, with multiprocessors, hyperthreading, real loads, etc., this is just "noise" in your program: no useful semantic value whatsoever. Passing in &i is of course a serious mistake, because the address points to somewhere on the stack which may or may not be valid at the point where the thread executes. So your basic program structure has a fundamental flaw here.

If you want to pass in the sequence number, you can pass it in as
AfxBeginThread(threadfunc, (LPVOID)i);

and in the threadfunc

```
UINT threadfunc(LPVOID p)
{
    UINT i = (UINT)p1
    ...
}
```

Derferencing i, which is what you are doing, will most likely give you gibberish. Any illusion that it is giving you a meaningful value is a complete accident, and will fail in any number of truly horrible ways.

See below...

On Sat, 29 May 2004 06:52:42 -0400, A. J. Rappaport
<please.reply.in.newsgroup.only@spamsucks.com> wrote:

```
>
>
>Good morning, Joseph and Scott. Thanks for your detailed and
>informative explanations.
>
>There is nothing at all critical about the timing of my tour threads,
>and the startup time also isn't particularly critical EXCEPT for one
```

```
>factor...
>
>I have a need that each thread knows the sequence in which its startup
>routine was called. So, I do this...
>
>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.

Had you written

```
int localSeq = *(int *)seq;
```

you would have had syntactically and semantically correct code as far as the pointer dereference is concerned. The fact that the pointer value you dereference is complete nonsense, of course, is another issue.

```
*****
>
> .
> .
> .
> (etc.)
>}
>
>
>Now, "seq", being a pointer, is subject to change (or, more
>accurately, the thing that it points to is subject to change) by the
>calling method (hence the need to make a local copy of seq as quickly
>as practical). Even so, if I call AfxBeginThread on the second thread
>before the first thread is up and running, the first thread will see
>the second thread's seq, and will think that it is the second thread.
>This same problem is there no matter what I pass AfxBeginThread (a
>pointer to a struct, etc.) for whatever reason ... for example, if I
>had to pass each thread a pointer to a serial port to listen on, then
>I don't want the thread getting a pointer to the wrong serial port.
*****
```

If you want to pass a pointer to a struct, you heap-allocate each struct, for example,

```
for(i = 0; i < n; i++)
{
int * p = new int;
...test for NULL
*p = i;
```

```
AfxBeginThread(threadfunc, *p)
}
```

will work because each reference is unique. Of course, you must do a 'delete' in the thread function after you are done with the value.

But if this is all you are passing, it seems silly, because a simple cast of the integer to an LPVOID and later back to an int (if you want safe compilation for 64-systems, you would use the type INT_PTR) is so much simpler. But if you need to pass complex information (more than a simple scalar) to the thread, then the allocation technique is the best way to handle this.

>

*>Hence the need for the "Sleep(x)" after the call to AfxBeginThread,
>to give each thread time to get going before calling the next one.*

One thing you can be certain of when you use Sleep() like this: your program is wrong. No other answer is valid. It is a simple heuristic, and I've never had it fail: a multithreaded system that will not work correctly without putting a Sleep() call in was never correct to begin with, and the fact that it is ever seen to work at all after the Sleep() is added is either sheerest coincidence, or a miracle. But it will eventually fail, probably spectacularly, probably for your best customer doing a demo to his or her management.

>

*>The value of "x" in Sleep(x) could be anything up to around 50 ms
>and it wouldn't really affect my program in any noticeable way. But,
>from what you guys are saying, even 50 ms may not necessarily always
>be enough. It would be quite bad news whenever some unusual delay
>happens so that the threads get the wrong arguments passed in.*

Pick any value for x. It will be wrong. You are applying sequential thinking to a parallel problem, and this won't work.

>

>Is there even an answer to this within the AfxBeginThread mechanism?

Since I've already described the solution that works, nothing else is needed

>

*>I suppose I could establish some public or file-scope flag that the
>calling function sets and the thread clears, and the calling function
>doesn't establish the next thread until the flag clears, or something
>like that?*

This would be a worse design than the Sleep(). If you need to sequence threads, you would use mutexes, queues, semaphores, events, or other proper synchronization mechanisms. But never spin-on-a-flag.

microsoft.public.vc.mfc: Re: AfxBeginThread startup times and overhead

Joseph M. Newcomer [MVP]

email: newcomer@flounder.com

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

MVP Tips: http://www.flounder.com/mvp_tips.htm