What is the best way to attach a debugger to a process in VC++ at just the right point in time?

Asked 15 years, 9 months ago Modified 4 years, 5 months ago

Viewed 21k times

21

When debugging, sometimes you need to attach an already running process instead of just starting the application in a debugger.

It's common for myself to put in a Sleep() or MessageBox call, so that it's easier to attach a debugger. I worry that some of these may be committed eventually to source control.

What is the best thing to do to avoid this situation while still delaying enough time so that you can attach your debugger to a running process?

Guarding the Sleep or message box with an `#ifdef _DEBUG` is one way, but I'm wondering if there is a better way.

With a Sleep you also have the problem that you may not attach in time. With a MessageBox you have the problem that you may be remotely debugging, or debugging a process that has no visible GUI (example running as a service on Vista) c++ debugging visual-c++ debugbreak

Share

Improve this question

Follow

edited Mar 19, 2009 at 19:19

Suma

34.3k ● 18 ● 129 ● 199

asked Mar 19, 2009 at 18:31

Brian R. Bondy

347k ● 126 ● 602 ● 640

Related: stackoverflow.com/questions/2789390/…
– DuckMaestro Apr 12, 2020 at 9:20

8 Answers

Sorted by: Highest score (default)

24

another variant, which I sometimes use is

```
while( !::IsDebuggerPresent() )
    ::Sleep( 100 ); // to avoid 100% CPU load
```

it should just silently wait until you attach your debugger to the process.

Share Improve this answer

Follow

answered Jan 26, 2016 at 15:51

Roman Kruglov

3,547 ● 3 ● 41 ● 48

11

✓

you can use DebugBreak, check these links:

http://www.epsilon-delta.net/articles/vc6_debug.html#breaking-with-debugbreak

http://blogs.msdn.com/calvin_hsia/archive/2006/08/25/724572.aspx

Share Improve this answer Follow

answered Mar 19, 2009 at 18:43

eglasius
36k ● 5 ● 68 ● 110

9

To attach a debugger at a particular point, you have several options:

The simplest is just to call `DebugBreak`, which is pretty much equivalent to `__asm int 3`, but also works on other architectures (MSVC for x64 doesn't allow inline assembly, if I recall correctly). This'll bring up the just-in-time debugger window, and you'll be able to select from registered debuggers (i.e. Visual Studio) to attach to the process.

Alternatively, you can introduce a call to `Sleep`, giving you an opportunity to attach the debugger. You should use `#ifdef _DEBUG` around this, to ensure that you don't actually ship with this code included.

One question: why can't you run the code from the IDE? Is it a service or an IIS-loaded DLL or similar?

In this case, you can check out the `ImageFileExecutionOptions` registry key, which allows you to attach a debugger at the moment that the process starts.

If you use cdb for this, you can configure it as either server or client to a WinDbg instance, and debug that way. I've done this in the past by using WinDbg as a kernel debugger, and by using ImageFileExecutionOptions to start `ntsd -d` with the named process. This causes WinDbg to break into user mode. This is sometimes a useful technique.

Share Improve this answer Follow

answered Mar 19, 2009 at 18:54

Roger Lipscombe
91.6k ● 59 ● 252 ● 396

Thanks, Sleep did the trick! – Rob W Oct 31, 2013 at 10:27

---

5 Since you're curious about why the code can't be run from the IDE, in my case it's because I'm trying to debug a DLL I've written that's injected into a process by a third party application, neither of which I directly control. The process the DLL is injected to isn't started by me (it's a child of a process I start) so I can't just attach the debugger to it either. And DebugBreak just causes the process to crash instead of showing the JIT debugger dialog. But `while (!IsDebuggerPresent()); DebugBreak();` seems to do the trick. – Cameron Nov 23, 2015 at 17:22

Freddy and Reoa have the correct solutions. But I wanted to add a reason why not to use a MessageBox.

7

Displaying a MessageBox only partially stops your application. Because you are showing UI, a message pump is still running on at least one thread in your program. So if your code does any of the following.

1. Communicates via Windows Messages
2. Has non-trivial UI
3. Is Multi-threaded

You will essentially be requesting a debugger in one state but attaching to your program in a completely different state. This can lead to bewildering situations and bugs.

We recently made a change in our code base to never show a MessageBox in order to facilitate break for this very reason. It produces very bad behavior for a non-trivial application.

Share Improve this answer Follow

answered Mar 19, 2009 at 18:52

JaredPar
753k ● 151 ● 1.3k ● 1.5k

2

Having to attach at 'just the right point' is a pain ... one option is to but explicit DebugBreak() statements into the code to force the issue, and guarding them with `#ifdef _DEBUG` would be a good idea. We use an ASSERT macro that can call DebugBreak(), so you can just write ASSERT(false)

Another option to consider is using 'image file execution options' to launch the debugger automatically. See this blog and the MSDN documentation.

Share Improve this answer Follow

answered Mar 19, 2009 at 18:44

Rob Walker
47.4k ● 15 ● 100 ● 137

2

If __debugbreak() or DebugBreak() will work for you, then I think that's probably the best approach.

However, I have run into some circumstances where for unknown reasons, __debugbreak() was just terminating the program immediately instead of waiting for a debugger to connect. I tried various ways to address this (registry hacks, etc.) but nothing seemed to work. (It may be relevant that in this case the process I was trying to debug was not started directly from a command line, but was instead started by a Java program.)

Anyway, In that case, I used this approach, which seemed to work well:

```
void waitForDebuggerToConnect() {
    #ifdef _DEBUG
    static volatile bool spin = true;
    while (spin)
    {}
    #endif
}
```

After I called this function, my program would hang indefinitely in the 'while' loop. I could then invoke the VC++ debugger and attach to the process. I could then use 'break all' to stop all threads. I could then find the stuck thread's call stack, and could use the debugger to inspect the top frame, and could inspect the value of 'spin' and then set it to 'false'. At this point, I could then step out of this function and/or set breakpoints and then resume some or all threads running.

Share Improve this answer Follow

answered Jul 9, 2020 at 19:17
Some Guy
509 ● 8 ● 17

1

Look up:

DebugBreak , __debugbreak and friends

or

static void timeToChase() { __asm { int 3; }; }

Share Improve this answer Follow

answered Mar 19, 2009 at 18:43
rama-jka toti
1,436 ● 10 ● 16

1

```
__asm int 3
```

This hard breakpoint will bring up the debug dialog, which will let you attach to the process. Wrap that in #ifdef _DEBUG and you'll only hit it in debug builds.

Share Improve this answer Follow

answered Mar 19, 2009 at 18:44

Aaron Saarela

4,006 ● 1 ● 20 ● 17