

JDK Deadlocks with 2 app domains inside same aspnet_wp.exe

Source: <http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.vjsharp/2004-04/0007.html>

From: Bruno Jouhier [MVP] (bjouhier_at_club-internet.fr)

Date: 04/02/04

Date: Fri, 2 Apr 2004 22:18:38 +0200

We have been experiencing some severe deadlocks in our ASP application since we migrated from J++ to J#.

I analyzed several thread dumps made while the server was in deadlock state, and they all seem to be happening inside the JDK. They also seem to be related to the fact that we are hosting 2 applications on the same server (so, we have 2 AppDomains inside the same aspnet_wp.exe process).

In one of the dumps, I found the following stack traces:

```
Thread 0x18ac R at Calendar::getInstance +0024
Thread 0x18ac Current State:Wait/Sleep/Join
0)* vjslib!java.util.Calendar::getInstance +0024 [no source information
available]
1) vjslib!com.ms.vjsharp.util.Rule::createCalendars +0018 [no source
information available]
2) vjslib!com.ms.vjsharp.util.Rule::getTimeForYearInRule +0015 [no source
information available]
3) vjslib!com.ms.vjsharp.util.Rule::compare +0012 [no source information
available]
4) vjslib!java.util.SimpleTimeZone::setEndRule +0061 [no source information
available]
5) vjslib!java.util.SimpleTimeZone::ctor +004d [no source information
available]
6) vjslib!java.util.TimeZone::__getTimeZone +009c [no source information
available]
7) vjslib!java.util.TimeZone::getTimeZone +015b [no source information
available]
8)
platform.bizlog!Platform.Bizlog.Currencies.PfCurrencyUpdater::_updateCurrenc
yInfo +01e1 [no source information available]
9) more frames from our server code ...
```

```
Thread 0x1a0c R at TimeZone::getDefault +0029
Thread 0x1a0c Current State:Wait/Sleep/Join
0)* vjslib!java.util.TimeZone::getDefault +0029 [no source information
```

available]

1) vjslib!java.util.Calendar::getInstance +0042 [no source information available]

2) vjslib!java.text.SimpleDateFormat::.ctor +0035 [no source information available]

3) vjslib!java.text.SimpleDateFormat::.ctor +0028 [no source information available]

4)

platform.bizlog!Platform.Bizlog.Currencies.PfCurrencyUpdater::_updateCurrencyInfo +01d8 [no source information available]

5) more frames from our server code ...

So:

in Thread 0x18ac, TimeZone.getTimeZone ends up calling Calendar.getInstance which waits

in Thread 0x1a0c, Calendar.getInstance calls TimeZone.getDefault which waits

I don't know how these classes are synchronized in J# but in the standard JDK, TimeZone.getTimeZone and TimeZone.getDefault are both synchronized on the TimeZone.class (they are static synchronized) and Calendar.getInstance is synchronized on Calendar.class (it is also static synchronized).

So, we have a deadlock (note: these 2 stack traces are the only ones that contain calls to Calendar and TimeZone so the waits cannot be explained by other threads that would have acquired locks on Calendar.class and TimeZone.class).

But the strangest thing is that the threads that generated these stack traces are in two different app domains (the PfCurrencyUpdater is a thread that we instantiate once in every application).

So, we have 2 threads from 2 different app domains that deadlock each other. So, I cannot even solve the deadlock by synchronizing on a global static object before making these calls, because each app domain will have its own static and my synchronization will only work inside one domain. Seems to me that I would have to go with a global Mutex to solve this, but this looks like terrible overkill. (or maybe synchronizing on Calendar.class before calling TimeZone.getTimeZone would fix it, but I have not tried it yet).

So, my question is the following: is it possible to have two threads from two app domains that deadlock each other inside the J# SDK? I thought that AppDomains guaranteed the right level of isolation and that this awful thing would not happen. But, unless I missed something obvious, this seems to happen.

At this point, my only explanation is that the app domains are sharing class-level locks in J#, which seems to violate app domains semantics.

Bruno.