

Re: Block in synchronized method

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.help/2005-02/1216.html>

From: Tony Dahlman (adahlman_at_jps.net)

Date: 02/25/05

Date: Fri, 25 Feb 2005 04:29:46 GMT

Tilman Bohn wrote:

> *[Some re-wrapping of quoted text ahead]*

>

> *In message <421C3EA4.3040109@jps.net>,*

> *Tony Dahlman wrote on Wed, 23 Feb 2005 08:28:23 GMT:*

>

> *[...]*

>

>> *Clearly I was on the wrong track thinking this was one of those*

>> *business "issues" with the tool and JVM producers. Curiosity is*

>> *undiminished, however. Any idea how much efficiency would be lost if*

>> *the Java spec didn't permit spurious wakeups?*

>

>

> *Sorry, no idea. I've never investigated it since the loop is the*

> *correct and robust way to do it anyway.*

>

> *[...]*

>

>> *seemed to require it some 7 to 10 years ago. In one case, however,*

>> *the while() was just checking if the application was ready to*

>> *complete. (There was no "done" variable for a thread, only an*

>> *"allDone" variable for the application.) Was that right or should I*

>> *rewrite it with a tighter loop? As I see it, a spurious wakeup would*

>> *do nothing....*

>

>

> *Looks ok to me at first glance, but I don't have time to look at it in*

> *detail right now, so don't depend on my word here. Come to think of it,*

> *don't depend on my word in any case. It's not like I haven't produced my*

> *share of deadlocks and livelocks in my time. :-)*

>

> *[...]*

>

>> *But I'm also curious about those "superflous condition broadcasts or*

>> *signals on the same condition". Where are those coming from in a Java*

>> *program that (rarely) might produce them?*

>

>
> *This means code executed in another thread that notifies threads*
> *waiting on that same lock, without having altered state to make the*
> *predicate hold. That could be either by accident (coding error), by*
> *malice (third party code), or possibly by design. In many cases bad*
> *design I suppose, but using coarse-grained monitors when you're waiting*
> *for different but supposedly somehow related conditions can sometimes*
> *make sense. For instance, maybe the notifying code can't be sure exactly*
> *which of those related predicates have changed at the time it notifies*
> *waiters. Then it would be natural to wake up all threads waiting on that*
> *lock under these different guards.*
>
>
>>*Is some of the underlying*
>>*code of the listener-event construct, which is such a plus for Java,*
>>*doing that, even if rarely?*
>
> [...]
>
> *Nah, just some other application code. You never know who might send*
> *you a signal, or when, even if you keep very tight wraps on your monitor*
> *(remember you can use any old Object as a monitor).*
>

Like,

```
Object lock = new Object();  
...  
synchronized(lock) {  
    ...  
}
```

...which is totally missing from, yet useable and maybe adviseable in, my code.

Your comments are much appreciated. But you are too humble about your status as an authority. Thanks for the work you've done understanding this at so many levels: convincing and instructive.

--Tony Dahlman