

Re: A little threading problem

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-12/0182.html>

From: Alban Hertroys (alban_at_magproductions.nl)

Date: 12/02/04

Date: Thu, 02 Dec 2004 11:41:54 +0100

To: Jeremy Jones <zanesdad@bellsouth.net>

Jeremy Jones wrote:

> *Alban Hertroys wrote:*

> *Notify is called before thread B (in this case) hits the*
> *condAllowed.wait() piece of code. So, it sits at that wait() for*
> *forever (because it doesn't get notified, because the notification*
> *already happened), waiting to be notified from the main thread, and the*
> *main thread is waiting on thread B (again, in this case) to call*
> *mainCond.notify(). This approach is a deadlock just wanting to happen*
> *(not waiting, because it already did happen). What is it exactly that*
> *you are trying to accomplish? I'm sure there is a better approach.*

Hmm, I already learned something I didn't know by reading through my own version of the output.

I added an extra counter, printed before waiting in the for-loop in Main. My wrong assumption was that acquire() would block other threads from acquiring until release() was called. In that case the for-loop would wait 3 times (once for each thread), which is what I want.

Unfortunately, in my output I see this:

T-A: acquire mainCond

...

T-B: acquire mainCond

...

T-B: released mainCond

...

T-A: released mainCond

Which is exactly what I was trying to prevent...

But even then, as you pointed out, there is still the possibility that one of the threads sends a notify() while the main loop isn't yet waiting, no matter how short the timespan is that it's not waiting...

As for what I'm trying to do; I'm trying to merge three huge XML files into single separate database records. Every database record contains related data from each of the XML files.

comp.lang.python: Re: A little threading problem

For practical purposes this is a two-stage process, where I first store an uncombined "record" from each XML file into the DB (marked as 'partial'), and then periodically merge the related records into one final record.

I could first store all data as 'partial' and then merge everything, but I consider it better to do this with relatively small batches at a time (queries are faster with smaller amounts of data, and the DB stays smaller too).

The reason I use threads for this is that (to my knowledge) it is not possible to pause an `xml.parsers.xmlproc.xmlproc.Application` object once it starts parsing XML, but I can pause a thread.

This is a timeline of what I'm trying to do:

```
Main start |combine XML |comb.
                |next batch |next
Application A run>.....*| |>.....*| | etc.
Application B run>.....*| |>.....*|
Application C run>.....*| |>.....*|
```

Legend:

```
> = thread is active
    * = batch ready, wait()
    | = timeline delimiter
```