

Re: python's threading has no "interrupt"?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2003-12/0291.html>

From: Jay O'Connor (jocconnor_at_cybermesa.com)

Date: 12/02/03

Date: Tue, 02 Dec 2003 10:15:58 -0700

Dave Brueck wrote:

>Jay wrote:

>

>

>>>You've noticed that there isn't an identical construct for what you were

>>>

>>>

>doing

>

>

>>>in Java, so it may be that the Python way will be a completely different

>>>approach to the problem rather than just a direct conversion from Java to

>>>Python syntax.

>>>

>>>

>>Well, I'll give an example from something I tried to do a few years ago

>>which I couldn't translate to Python, though I took a stab at it.

>>

>>The language in question was a VM language with it's own internal

>>process model. The process model was all internal to on eOS process to

>>it was a cooperative model of multi-threading

>>

>>The application I wrote was a web server on the front of an application

>>so that web requests were all handled in the application. IOW, the

>>application had a web server interface)

>>

>>The language process model had both the ability to set process

>>priorities, as well as allow processes to sleep and cede control to

>>other processes.

>>

>>When a web request would come in, the handling of the request would be

>>forked as a separate process so that the server could accept the next

>>request. Both the server process and the handling process(es) were at

>>the same priority level so in theory each process would run to

>>completion before allowing another process to run. For this reason,

>>both the server process and the handling processes would issue a 'yield'

>>at periodic strategic points, allowing themselves to temporarily halt

comp.lang.python: Re: python's threading has no "interrupt"?

>>*and for the next process of equal priority that was waiting to run.*

>>

>>

>

>*So, if I understand this correctly, this amounted to a manual task switcher,
>right? (as no more than one job was allowed to run concurrently). Was this
>really the desired behavior? If each process halts while the others run and
>does so only to prevent starvation in the other processes, wouldn't you get
>more or less the same results by just using normal forking/threading wherein
>the OS ensures that each process/thread gets a slice of the CPU pie? (IOW,
>unless there was some other reason, it seems that the explicit yielding was to
>work around the constraints of the process model)*

>

>

Partially, the processes were sharing a (sometimes very large) common in-memory object web so forking separate OS level threads was not feasible. Also, the server ran on multiple OS platforms so OS level tricks or threading mechanisms were not viable
So... what was going wrong that warranted killing the process in the first

>*In practice the need for that is pretty rare, especially for a server,
>the main case coming to mind being the execution of user-supplied code (which
>is pretty scary in and of itself!).*

>

It didn't happen that often but it was mostly due to the client disconnecting prematurely and when your server is handling heavy traffic, it does become a resource issue after awhile. Like I said in another post, there was also bookkeeping down during this time as well.