

# python threading and timing

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-10/msg00697.html>

---

- *From:* "Oeyvind Brandtsegg" <[obrandts@xxxxxxxxxx](mailto:obrandts@xxxxxxxxxx)>
  - *Date:* Sun, 1 Oct 2006 22:28:10 +0200
- 

hello

I'm writing a Python application for live music performance/improvisation, using csound as the synthesis engine, and Python for compositional algorithms, GUI and control.

I creating several threads:

- one for GUI
- one for csound,
- one for a timed queue (timed automation events),
- one for communication with a tcp based sensor interface.

My threads do not share data,

I use threads merely to enable several processes to run in paralell.  
I think I do not need to use locks to control threads in this case,  
but I'm not sure (?)

Also, I wonder what method I should be using to get a precise timing in my automation thread (acting as a sequencer).

I've attached two simple examples of different ways I could implement threads.

The only task in these examples is to generate timed counters, but the general procedure would be the same for more complex timed tasks. One example uses `time.sleep()` to release a thread's interpreter lock, and at the same time provide the time interval until the next iteration.

The other example uses a thread event `wait()` call.

It seems that the example using `wait()` does not provide the same precision as `time.sleep()` does (the counters does not sync, e.g. a counter with a time interval of 0.1 does not count 10 times as fast as a counter with a time interval of 1 second).

Any insights on how to do this the best way (thread safety and time precision) would be greatly appreciated.

best

Oeyvind Brandtsegg

# test of threading routines for calling functions periodically

## python threading and timing

```
import threading
import time

class CounterTest(threading.Thread):
    def __init__(self, res, name):
        self.name = name
        self.counter = 0
        self.keepRunning = 1
        self.res = res
        threading.Thread.__init__(self)
    def run(self):
        while self.keepRunning:
            print self.name, self.counter
            self.counter += 1
            time.sleep(self.res)
    def stopme(self):
        self.keepRunning = 0

c1 = CounterTest(1,'one')
c2 = CounterTest(0.1,'two')

print 'starting threads'
c1.start()
c2.start()

#allow threads to run for a while
time.sleep(5)
c1.stopme()
c2.stopme()

print 'threads stopped'# test of threading routines for calling functions periodically

import threading
import time

class CounterTest(threading.Thread):
    def __init__(self, res, name):
        self.name = name
        self.counter = 0
        self._stopevent = threading.Event()
        self._sleepperiod = res
        threading.Thread.__init__(self)
    def run(self):
        while not self._stopevent.isSet():
            print self.name, self.counter
            self.counter += 1
            self._stopevent.wait(self._sleepperiod)
    def stopme(self):
        self._stopevent.set()
```

## python threading and timing

```
c1 = CounterTest(1,'one')
c2 = CounterTest(0.1,'two')

print 'starting threads'
c1.start()
c2.start()

#allow threads to run for a while
time.sleep(5)
c1.stopme()
c2.stopme()

print 'threads stopped'
```