Notify the main thread when a thread is done

Asked 12 years, 6 months ago Modified 12 years, 6 months ago

Viewed 3k times

2

I am fairly new to Python programming and Threads isn't my area of expertise. I have a problem for which i would hope that people here can help me out with.

Task: as a part of my master thesis, i need to make a mixed reality game which involves multiplayer capability. in my game design, each player can set a bunch of traps, each of which is active for a specific time period e.g. 30 secs. In order to maintain a consistent game state across all the players, all the time check needs to be done on the server side, which is implemented in Python.

I decided to start a python thread, everytime a new trap is laid by a player and run a timer on the thread. All this part is fine, but the real problem arises when i need to notify the main thread that the time is up for this particular trap, so that i can communicate the same to the client (android device).

i tried creating a queue and inserting information into the queue when the task is done, but i cant do a queue.join() since it will put the main thread on hold till the task is done and this is not what i need nor is it ideal in my case,

since the main thread is constantly communicating with the client and if it is halted, then all the communication with the players will come to a standstill.

I need the secondary thread, which is running a timer, to tell the main thread, as soon as the time runs out that the time has run out and send the ID of the trap, so that i can pass this information to the android client to remove it. How can i achieve this ??

Any other suggestions on how this task can be achieved without starting a gazillion threads, are also welcome.. :) :)

Thanks in advance for the help..

Cheers

python **multithreading**

Share

Improve this question

Follow

asked Jun 18, 2012 at 9:08

vivek86

745 ● 1 ● 10 ● 20

3 Answers

Sorted by: Highest score (default)

3

i have finally found a nice little task scheduler written in python, which actually is quite light and quite handy to schedule events for a later time or date with a callback

mechanism, which allows the child thread to pass-back a value to the main thread notifying the main thread of its status and whether the job was successfully done or not.

people out there, who need a similar functionality as the one in the question and dont want to haggle around with threads can use this scheduler to schedule their events and get a callback when the event is done

here is the link to [APScheduler]

Share Improve this answer

Follow

answered Jun 21, 2012 at 12:18

vivek86

745 ● 1 ● 10 ● 20

Hi this APScheduler doesn't come back to the main thread for me... some code of how you did it would be nice – Magoo Aug 9, 2022 at 3:25

1

It may be easier to have the timers all done in the main thread - have a list of timers that you keep appending new ones to. Each timer doesn't actually *do* anything, it just has a time when it goes off - which is easier if you work in arbitrary 'rounds' than in real time, but still doable. Each interval, the mainloop should check all of them, and see if it is time (or past time) for them to expire - if it is, remove them from the list (of course, be careful about removing items from a list you're iterating over - it mightn't do what you expect).

If you have a *lot* of timers, and by profiling you find out that running through all of them every interval is costing you too much time, a simple optimisation would be to keep them in a heapq - this will keep them sorted for you, so you know after the first one that hasn't expired yet that none of the rest have either. Something like:

```
while True:
    if not q:
        break
    timer = heapq.heappop(q)
    if timer.expiry <= currenttime:
        # trigger events
    else:
        heapq.heappush(q)
        break
```

This does still cost you one unnecessary pop/push pair, but its hard to see how you would do better - again, doing something like:

```
for timer in q:
    if timer.expiry <= currenttime:
        heapq.heappop(timer)
        # trigger events
    else:
        break
```

Can have subtle bugs because list iterators (functions in `heapq` work on sequences and use side effects, rather than there being a full-fledged `heapq` class for some reason) work by keeping track of what index they're up to - so if you remove the current element, you push

everything after it one index to the left and end up skipping the next one.

The only important thing is that `currenttime` is consistently updated each interval in the main loop (or, if your heart is set on having it in real time, based on the system clock), and `timer.expiry` is measured in the same units - if you have a concept of 'rounds', and a trap lasts six rounds, when it is placed you would do `heapq.heappush(q, Timer(expiry=currenttime+6)`.

If you do want to do it the multithreaded way, your way of having a producer/consumer queue for cleanup will work - you just need to not use `Queue.join()`. Instead, as the timer in a thread runs out, it calls `q.put()`, and then dies. The mainloop would use `q.get(False)`, which will avoid blocking, or else `q.get(True, 0.1)` which will block for at most 0.1 seconds - the timeout can be any positive number; tune it carefully for the best tradeoff between blocking long enough that clients notice and having events go off late because they only just missed being in the queue on time.

Share Improve this answer

Follow

answered Jun 18, 2012 at 9:44

lvc

35k ● 10 ● 75 ● 99

thanks for the quick reply.. i will definitely have a look at the Heap idea and see if i can use it in my scenario. I have one question though about the idea for the multi-threaded way:
– vivek86 Jun 18, 2012 at 11:26

when should i precisely do q.get() ?? since all the threads are running simultaneously, do i need to write an infinite loop in the main thread, constantly checking all the threads and whether they are alive or dead and then get the information from them ?? what i wanted or rather had in mind was something of a sort of a notification to the main thread. For e.g. suppose there are 10 threads running, other than the main thread all started at different times and with different time durations. and suppose thread_4's timer just ran out, it should notify the main thread that its time is up. – vivek86 Jun 18, 2012 at 12:18

1 @vivek86 the queue is a producer-consumer queue shared between all threads. The worker threads are the producers - they call `q.put(timer_info)` when their timer runs out. That is, the queue is your notification mechanism. The main thread would have a very similar loop to the single-thread case I described - it takes everything off the queue in turn and deals with it. It doesn't need to check every thread, or even know how many threads are running - once a thread has sent its notification via the queue, it has nothing else to do and can exit (unless you reuse them for efficiency). – lvc Jun 18, 2012 at 12:30

0

The main thread creates a queue and a bunch of worker threads that are pulling tasks from the queue. As long as the queue is empty all worker threads block and do nothing. When a task is put into the queue a random worker thread acquires the task, does it job and sleeps as soon as its ready. That way you can reuse a thread over and over again without creating a new worker threads.

When you need to stop the threads you put a kill object into the queue that tells the thread to shut down instead

of blocking on the queue.

Share Improve this answer

Follow

answered Jun 18, 2012 at 9:53

Vivek S

5,540 ● 8 ● 55 ● 74

The problem with using a fixed number of worker threads (and, indeed, the problem with threads in general for this) is that OP most likely cares about the timers starting at the right time. Worker threads reading from a work queue makes the most sense when you don't care about *when* things start, just that they do start once there's a worker available. – lvc Jun 18, 2012 at 13:18