

Re: Converting existing module/objects to threads

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-10/msg03661.html>

- *From:* "martdi" <martin.dion@xxxxxxxxxx>
 - *Date:* 18 Oct 2006 20:33:15 -0700
-

I am not sure if I understand you question well, but:

in the `__init__` of the thread subclass, you can instantiate an object of the class that makes the work

or

ControllerThread could extend both classes and i don't think there would be a problem.

Problems in multithreading usually happen when many threads try to access the same ressource at the same time or that one thread waits for an other thread to finish, and that other thread waits for the first one to finish. The Queue module is threadsafe and it uses locking to prevent multiple threads to access it at the same time. Using Queue is a lot easier than having to manage locks by yourself.

As long as you do not share data between your classes like static members, database connections, or I/O on the same ressource, you probably won't encounter problems.

jdlists@xxxxxxxxxx wrote:

martdi wrote:

jdlists@xxxxxxxxxx wrote:

I have inherited some existing code, that i will explain in a moment, have needed to extend and ultimately should be able to run in threads. I've done a bunch of work with python but very little with threads and am looking for some pointers on how to implement, and if the lower level modules/objects need to be rewritten to use `threading.local` for

Re: Converting existing module/objects to threads

all local variables.

I have a module that communicates with a hardware device, which reads data off of sensors, that can only talk with one controller at a time.

The controller (my module) needs to (in its simplest form) init, configure the device, request data, and write out xml, sleep, repeat.

The new request is that the device needs to be queried until a condition is true, and then start requesting data. So an instance of a controller needs to be dedicated to a hardware device forever, or until the program ends....which ever comes first.

This currently works in a non-threaded version, but only for one device at a time, there is a need to create a single windows(yeach) service that talks to many of these devices at once. I don't need worker threads that handle seperate portions of the entire job, i need a single application to spawn multiple processes to run through the entire communication from configure to report, sleep until the next interval time and run again. The communication could last from 1 minute to 10 minutes before it ends.

Here is the code layout in pseudocode.

```
module.Object – controller.Main – handles all socket communications
```

```
class subcontroller(controller.Main):
def __init__(self,id,configurationFile):
controller.Main.__init__(self)
// instantiate variables and local objects that handle
configuration, logic and data output

def configure(self,configurationFile):
//read configurationFile and configure device

def process(self):
while 1:
```

Re: Converting existing module/objects to threads

//based on configuration file, query the device until condition is true and then write xml, sleep until time to repeat and run again.

within controller there are 5 objects and subcontroller is a single object that loads other objects from the inherited controller.System

I'm trying to figure out how difficult it is going to be to convert this to a threaded application. The original controller.Main is built to talk to devices in series, never in parallel. so no objects are considered to be thread safe, but no instance of any of the objects should need to share resources with any other instance of the same object. they would all have unique configuration files and talk to devices on unique ip/ports.

on a unix system, forking, while potentially not optimal, would be a fine solution, unfortunately this needs to run on windows.

I know i have left out many details, but hopefully this is enough to at least enable some kind of solution to lend an opinion or two.

many thanks
jd

Taking a look at asyncore could be worthwhile, but if you want to implement it with threads, you may be able to do it this way:

In your main file, from where you start the program, let's call it main:

```
main(self)
Load Required configuration
spawn threads (1 for each controller)
define a queue object from module queue.queue used for communication with threads
enter an infinite loop that checks for the conditions
once conditions are met, notify the proper thread
```

```
class ControllerThread(threading.Thread):
def __init__(self):
```

Re: Converting existing module/objects to threads

define a queue here, to process messages from the main
call `threading.Thread.__init__(self)`

define method to load config for thread objects
(you might want to pass an argument to `init` to load your configs
from a file)
(you might also want to pass the queue of the main program to the
thread to send it messages)
define methods to post messages to the queue like `read`, `send` to the
machine, `stop`, ...

define the `run` method that is what will be called when you start
your thread.
this method should enter an infinite loop that will check if
something has to be done (check in the queue).

hope this might help you
good luck

thanks for the comments. the `ControllerThread` already extends a class.
does this cause problems with classes that must extent
`threading.Thread`. Normally it should not matter, but with threads i'm
unsure. Should i just instantiate the object that i'm normally
extending in the `ControllerThread.__init__`, and call it from the
`self.classthatusedtoextend.method()`, or does it not matter.

again, thanks.