QGIS Application - Bug report #20711
QgsTask and task manager cause crash when used in function scope

2018-12-04 01:23 AM - Graham Duls

Status:	Open		
Priority:	High		
Assignee:			
Category:	Python bindings / sipify		
Affected QGIS version:	3.4.2	**Regression?:**	No
Operating System:	Windows 10 64-bit	**Easy fix?:**	No
Pull Request or Patch supplied:	No	**Resolution:**	
Crashes QGIS or corrupts data:	Yes	**Copied to github as #:**	28531

Description

I have a detailed description of the issue in an answer on the stack:
https://gis.stackexchange.com/a/304801/89589

Crash ID: 5aa220a1a8bf05838b9ee7e715ec57d7c74f0b05

Stack Trace

CPLStringList::empty :
PyInit__core :
QgsTask::start :
QThreadPoolPrivate::reset :
QThread::start :
BaseThreadInitThunk :
RtlUserThreadStart :

QGIS Info
QGIS Version: 3.4.2-Madeira
QGIS code revision: 22034aa070
Compiled against Qt: 5.11.2
Running against Qt: 5.11.2
Compiled against GDAL: 2.3.2
Running against GDAL: 2.3.2

System Info
CPU Type: x86_64
Kernel Type: winnt
Kernel Version: 10.0.17134

History

#1 - 2018-12-05 05:55 PM - Giovanni Manghi

- Priority changed from Normal to High

- Status changed from Open to Feedback

- Category changed from PyQGIS Console to Python bindings / sipify

Please add the description also here, thanks.

#2 - 2018-12-06 12:02 AM - Graham Duls

QGIS version 3.4.2 If you define a task and add it to the task manager within a function scope, it will cause the task to either not execute properly or, at worst, crash QGIS. Consider the following code:

```
import random
from time import sleep
from qgis.core import QgsApplication, QgsTask, QgsMessageLog, Qgis

MESSAGE_CATEGORY = 'Wasting time'

def do_task(task, wait_time):
    """
    Raises an exception to abort the task.
    Returns a result if success.
    The result will be passed together with the exception (None in
    the case of success) to the on_finished method
    """
    QgsMessageLog.logMessage('Started task {}'.format(task.description()),
                             MESSAGE_CATEGORY, Qgis.Info)
    wait_time = wait_time / 100
    total = 0
    iterations = 0
    for i in range(100):
        sleep(wait_time)
        # use task.setProgress to report progress
        task.setProgress(i)
        arandominteger = random.randint(0, 500)
        total += arandominteger
        iterations += 1
        # check task.isCanceled() to handle cancellation
        if task.isCanceled():
            stopped(task)
            return None
        # raise an exception to abort the task
        if arandominteger == 42:
            raise Exception('bad value!')
    return {'total': total, 'iterations': iterations,
            'task': task.description()}

def stopped(task):
    QgsMessageLog.logMessage(
        'Task "{name}" was canceled'.format(
            name=task.description()),
        MESSAGE_CATEGORY, Qgis.Info)

def completed(exception, result=None):
    """This is called when do_task is finished.
    Exception is not None if do_task raises an exception.
    Result is the return value of do_task."""
    if exception is None:
        if result is None:
            QgsMessageLog.logMessage(
```

```
            'Completed with no exception and no result '\
            '(probably manually canceled by the user)',
            MESSAGE_CATEGORY, Qgis.Warning)
        else:
          QgsMessageLog.logMessage(
            'Task {name} completed\n'
            'Total: {total} ( with {iterations} '
            'iterations)'.format(
              name=result['task'],
              total=result['total'],
              iterations=result['iterations']),
            MESSAGE_CATEGORY, Qgis.Info)
    else:
      QgsMessageLog.logMessage("Exception: {}".format(exception),
                               MESSAGE_CATEGORY, Qgis.Critical)
      raise exception

  # Create and execute a few tasks
  task1 = QgsTask.fromFunction(u'Waste cpu 1', do_task,
                               on_finished=completed, wait_time=4)
  task2 = QgsTask.fromFunction(u'Waste cpu 2', do_task,
                               on_finished=completed, wait_time=3)
  QgsApplication.taskManager().addTask(task1)
  QgsApplication.taskManager().addTask(task2)
```

If you put this in the script editor in QGIS and execute it, it will run just fine. The user can cancel. The exceptions are raised, occasionally. All is well. Now if you replace the last block of the code to be within a function scope, like so:

```
  def task_create_and_execute():
    # Create and execute a few tasks
    task1 = QgsTask.fromFunction(u'Waste cpu 1', do_task,
                                 on_finished=completed, wait_time=4)
    task2 = QgsTask.fromFunction(u'Waste cpu 2', do_task,
                                 on_finished=completed, wait_time=3)
    QgsApplication.taskManager().addTask(task1)
    QgsApplication.taskManager().addTask(task2)

  task_create_and_execute()
```

Now when you run the script you will find that either task1 will appear to run, but the completed() function will never be reached (task2 will NOT run) or QGIS will crash. If you run it twice, QGIS will most surely crash. This will also happen when using the class method.

#3 - 2018-12-06 09:37 AM - Giovanni Manghi

- Status changed from Feedback to Open