TRACING PYTHON MEMORY LEAKS

By: Marek Majkowski on November 14, 2008

LShift Technology

While I was writing a python daemon, I noticed that my application process memory usage is growing over time. The data wasn't increasing so there must have been some memory leak.

It's not so easy for a Python application to leak memory. Usually there are three scenarios:

1. some low level C library is leaking
2. your Python code have global lists or dicts that grow over time, and you forgot to remove the objects after use
3. there are some reference cycles in your app

I remembered the post from Marius Gedminas, in which he traced his memory leaks, but I haven't noticed before that he published his tools.
The tools are awesome. Just take a look at my session:

```
$ pdb ./myserver.py
> /server.py(12)()
-> import sys
(Pdb) r
```

```
2008-11-13 23:15:36,619 server.py       INFO   Running
with verbosity 10 (>=DEBUG)
2008-11-13 23:15:36,620 server.py       INFO   Main
dir='./server', args=[]
```

After some time, when my application collected some garbages I pressed Ctrl+C:

```
2008-11-13 18:41:40,136 server.py       INFO   Quitting
(Pdb) import gc
(Pdb) gc.collect()
58
(Pdb) gc.collect()
0
```

Let's see some statistics of object types in memory:

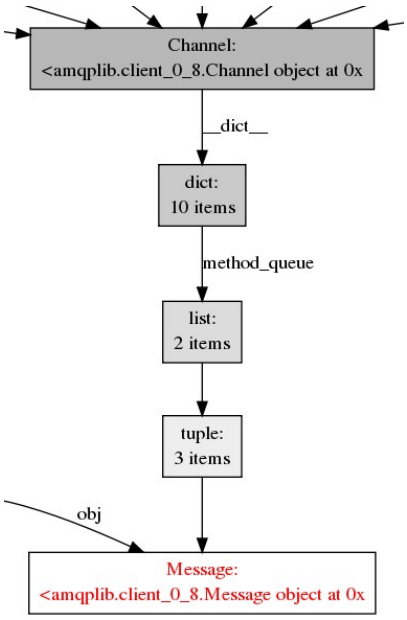
```
(Pdb) import objgraph
(Pdb) objgraph.show_most_common_types(limit=20)
dict                        378631
list                        184791
builtin_function_or_method 57542
tuple                       55478
Message                     48129
function                    45575
instancemethod              31949
NonBlockingSocket           31876
NonBlockingConnection       31876
_socketobject               31876
_Condition                  28320
AMQPReader                  14900
cell                        9678
```

Message objects definitely shouldn't be in the memory. Let's see where are they referenced:

```
(Pdb) objgraph.by_type('Message')[1]
<amqplib.client_0_8.Message object at 0x8a5b7ac>
(Pdb) import random
```

```
(Pdb) obj = objgraph.by_type('Message')
[random.randint(0,48000)]
(Pdb) objgraph.show_backrefs([obj], max_depth=10)
Graph written to objects.dot (15 nodes)
Image generated as objects.png
```

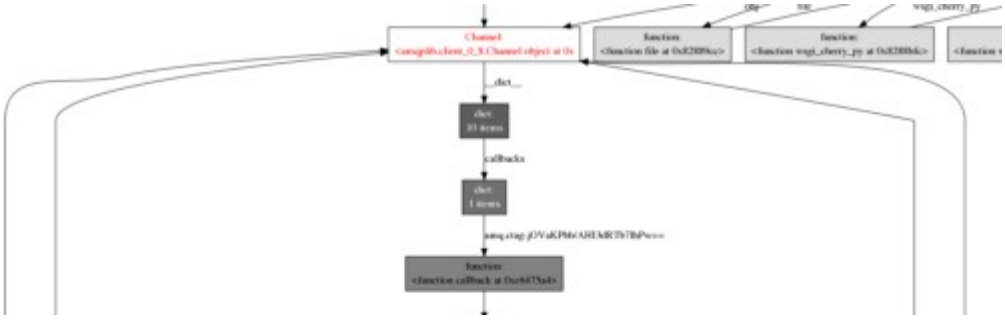
This is what I saw:



Ok. A Channelobject still has references to our Message. Let's move on to see why Channel is not freed:

```
(Pdb) obj = objgraph.by_type('Channel')
[random.randint(0,31000)]
(Pdb) objgraph.show_backrefs([obj], max_depth=10)
Graph written to objects.dot (35 nodes)
Image generated as objects.png
```

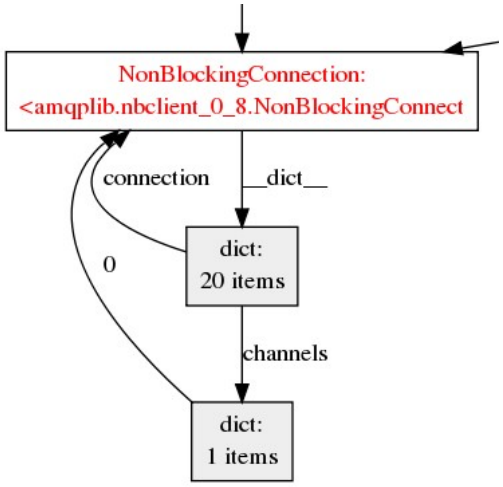
Channel object references are much more interesting – we just caught a reference cycle here!



There is also one other class that's not being freed – NonBlockingConnection:

```
(Pdb) obj = objgraph.by_type('NonBlockingConnection')
[random.randint(0,31000)]
(Pdb) objgraph.show_backrefs([obj], max_depth=10)
Graph written to objects.dot (135 nodes)
Image generated as objects.png
```

Here's the cycle we're looking for:



To fix this issue it's enough to break the reference loops in one place. This is the code that fixes the reference loops:

```
# we don't need channel and connection any more
channel.close()
connection.close()
# remove the reference cycles:
del channel.callbacks
del connection.channels
del connonection.connection
```

26 COMMENTS

Noel Welsh
November 14, 2008 at 3:58 pm

Isn't this case a result of Python lacking a real garbage collector? Cycles should prevent garbage being collected.

Kevin
November 14, 2008 at 4:19 pm

The garbage collector will collect objects in reference cycles unless they have del — make sure to see http://docs.python.org/library/gc.html#gc.garbage

Seun Osewa
November 14, 2008 at 6:06 pm

I think the tools you've demonstrated are interesting, but why should one not just use gc.collect() from time to time to remove the reference cycles?

Hazka
November 14, 2008 at 8:14 pm

I might have missed a step here but how did the Channel object graph lead you to the cycle in NonBlockingConnection?

Marius Gedminas
November 14, 2008 at 11:05 pm

I'm happy you found my tools useful!

The thing with the reference cycle is strange — Python can collect those just fine, as long as the objects in the cycle don't have **`del`** methods. Do they? I believe my latest version of objgraph.py is supposed to highlight objects with **`del`** in some way in the output graph.

By the way I suggest using `random.choice(objgraph.by_type('Foo'))` instead of indexing with a random index from a range that you have to manually specify.

Marius Gedminas
November 14, 2008 at 11:06 pm

Just a clarification: that bold-ed ‘del’ should really be ‘underscode underscore del underscore underscore’. I don’t know how to get past this blog markup, and because there’s no “Preview” button I don’t want to experiment.

marek
November 15, 2008 at 3:06 pm

Marius Gedminas:

> Python can collect those just fine, as long as the objects in the cycle donâ€™t have del methods. Do they?

Kevin:

> The garbage collector will collect objects in reference cycles unless they have del â€” make sure to see http://docs.python.org/library/gc.html#gc.garbage

From the python docs:

> Objects that have del() methods and are part of a reference cycle cause the entire reference cycle to be uncollectable.

From my application:

```
(Pdb) getattr(random.choice(objgraph.by_type('Channel')),
'__del__')
<bound method Channel.__del__ of <amqplib.client_0_8.Channel object
at 0xbbba7cc>>
```

In general python should do fine with collecting cycles. In my case the cycles are coming from an external library, which is not perfect and uses del on Channel objects. I agree that the problem with comes from the broken library rather than python garbage collector. But it’s nice to be able to see the cycles anyway.

Seun Osewa:

> I think the tools youâ€™ve demonstrated are interesting, but why should one not just use gc.collect() from time to time to remove the reference cycles?

I did it just at the beginning.

Hazka

> I might have missed a step here but how did the Channel object graph lead you to the cycle in NonBlockingConnection?

It didn’t. Both *Message* and *NonBlockingConnection* are taken from the

objgraph.showmostcommon_types output.

Chui Tey
November 28, 2008 at 10:18 am

Marius, objgraph.py rocks. I had to use it to troubleshoot leaks at work over the past two days, and it's a time saver.

Ulrik
March 1, 2009 at 4:57 pm

I'm looking at my toy application and its memory usage is constantly growing — very fast if I put the catalog rescan rate to 1/sec (normally 1/hour). However I can't see any significant change in the "most used objects" graph and specifically the overview over objects from my objects show no exaggerated numbers at all — they are constant.

It is safe to conclude that some library I use, probably in C, (like gnomedesktop) is leaky?

marek
March 2, 2009 at 10:48 pm

> Ulrik: It is safe to conclude that some library I use, probably in C, (like gnomedesktop) is leaky?

Try to use Valgrind.

Wil Tan
September 14, 2009 at 10:36 am

Wow! The tools are cool but I didn't have to use them at all because thanks to this blog entry I know my problem is due to the amqplib that we're using!

Thanks so much (and also for RabbitMQ)!

Gabriel Faure
May 24, 2010 at 1:59 am

Used this successfully in two projects. Awesome stuff!

Niklas
November 12, 2010 at 8:04 am

This helped me getting started with python memory leaks and objgraph, thank you for that!

Josh
August 19, 2011 at 3:46 am

Three years later and this still helped me. Thanks for keeping this up.

Robin Wittler
March 29, 2013 at 5:18 pm

It is 2013 and your article is actual as it was on 2008. Thx.
Helped me alot.

cheers,
Robin

Bhupesh
July 25, 2016 at 4:29 am

Hi,

1. I have a situation wherein my application is leaking some memory after significant amount of time. In such scenarios how can I debug it? How can I let the application run in pdb mode for sometime until it hits a trigger?
2. Can you also point out the dependency needed for building all the graphs. I tried to figure out but I am not able to find all the dependency for my CentOS.

Please help

LEAVE A REPLY

Your email address will not be published.

Your comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong>`

Name *

Email *

URL

Post Comment

CATEGORIES

- Agile
- Android
- APIs
- Bigwig
- C
- C#
- Case Studies
- Clojure
- Cloud
- Cryptology
- Debian
- Delphi
- Elm
- Erlang
- F#
- FPGA
- Free and Open Source Software
- git
- Go
- Hardware
- Haskell
- Howto
- Humour
- iOS
- Java
- Javascript
- Kotlin
- Labs
- LShift
- MacOSX
- mercurial-server
- mods
- Natural Language
- Networking
- Node.js
- Operations
- Our Software
- Politics
- Pontification
- Programming
- Project Management

Python
RabbitMQ
Rant
Reflection
Reviews
Ruby
Rust
Scala
Security
Services
Smalltalk
SQL
Standards
Technology
Tool chains
Tools
Ubuntu
Uncategorized
User Experience
Version control
Water cooler
Web
Windows

FEEDS

Blog posts