Let's Hunt a Memory Leak

Sanket Patel

Site Reliability Engineer @ LinkedIn

Agenda

Explore:

- **Python and Objects**
  - Allocs
  - Deallocs
- **The Leak**
  - with confirmed existence
- **Hunting!**

Won’t Explore:

- Specifics of memory management and garbage collection
- Why did *Katappa* kill *Bahubali*

Python 2 and Integers

```
Python 2.7.15 (default, Jan 12 2019, 21:07:57)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> import os, psutil, gc, time


>>> l=[i for i in range(100000000)]
>>> print(psutil.Process(os.getpid()).memory_info())
pmem(rss=3244871680L, vms=7824240640L, pfaults=1365384, pageins=460)
```

Python 2 and Integers

```
Python 2.7.15 (default, Jan 12 2019, 21:07:57)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> import os, psutil, gc, time

>>> l=[i for i in range(100000000)]
>>> print(psutil.Process(os.getpid()).memory_info())
pmem(rss=3244871680L, vms=7824240640L, pfaults=1365384, pageins=460)

>>> del l
```

Python 2 and Integers

```
Python 2.7.15 (default, Jan 12 2019, 21:07:57)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import os, psutil, gc, time

>>> l=[i for i in range(100000000)]
>>> print(psutil.Process(os.getpid()).memory_info())
pmem(rss=3244871680L, vms=7824240640L, pfaults=1365384, pageins=460)

>>> del l
>>> print(psutil.Process(os.getpid()).memory_info())
pmem(rss=2509352960L, vms=6964514816L, pfaults=1381131, pageins=460)
```

Objects

THEY SAY

Everything in Python is an Object

What is it?

```
typedef struct _object {
    PyObject_HEAD      // contains ref_count
} PyObject;
```

What is it?

```
typedef struct _object {
    PyObject_HEAD      // contains ref_count
} PyObject;


typedef struct {
    PyObject_HEAD
    long ob_ival;
} PyIntObject;
```

Free Lists

```
static PyListObject *free_list[PyList_MAXFREELIST];
static PyDictObject *free_list[PyDict_MAXFREELIST];
```

Free Lists

```
static PyListObject *free_list[PyList_MAXFREELIST];
static PyDictObject *free_list[PyDict_MAXFREELIST];

static PyFrameObject *free_list = NULL;
static int numfree = 0;          /* number of frames currently in free_list */
#define PyFrame_MAXFREELIST 200
```

Garbage Collection

- Generation based
  - A linked list for each generation
- When run in last generation (= 2)
  - Clear free lists as well

```
/* Clear free list only during the collection of the highest
 * generation */
if (generation == NUM_GENERATIONS-1) {
    clear_freelists();
}
```

Allocation

ref_count++

1. If free_list has space
    - Use last slot from free_list
2. Else: allocate memory
    - Initialize the object
    - Register object with GC (ie: add in linked list)*

*

not every object gets GC tracked

1. Immutables
   - long, float, string
2. containers
   - list, dict

Deallocation

when ref_count drops to 0

1. Untrack from GC (ie: modify pointers)*
2. Decrement ref count of contained types*
3. If: free_list has space
    - append to free_list
4. Else: free_list is full
    - free the memory

Recap

1. free_list
2. Generational GC
3. Alloc
    - ref_cnt++
    - free_list or alloc
    - GC tracking*
4. Dealloc
    - ref_cnt == 0
    - GC untrack
    - free_list or dealloc

Python 2 and Integers

free_list: **UNLIMITED** IN SIZE

Why?

- SPEED!
- Allocate space for N(=24) objects in one go.
- Deallocate never (except GC in gen 2)

https://superuser.blog/python-2-integers/

The Leak

A Good ‘ol Flask App

What does it do?

- Serve metrics for services

What does it contain?

- Fake metrics generator
- Helper which converts response in JSON
- Performance profiling
- An amplified leak!

The App

```
from flask import Flask
from helpers import _get_service_metrics, json_response

app = Flask('pycon')

@app.route('/metrics/<service>')
@json_response
def get_service_metrics(service):
    # returns list of metric data points
    data = _get_service_metrics(service)
    return data
```

Memory Usage

Initial Memory Usage:

```
~/workspace/pycon $ ./mem_usage.sh app.py
24MB
```

Sending *Some* Traffic:

```
/workspace/pycon $ ab -n50 http://127.0.0.1:5000/metrics/test_service
```

Memory Usage After:

```
~/workspace/pycon $ ./mem_usage.sh app.py
4085MB
```

The App

with added GC

```
import gc
from flask import Flask
from helpers import _get_service_metrics, json_response


app = Flask('pycon')


@app.route('/metrics/<service>')
@json_response
def get_service_metrics(service):
    # returns list of metric data points
    data = _get_service_metrics(service)
    gc.collect()
    return data
```

Memory Usage

with added GC

Initial Memory Usage:

```
~/workspace/pycon $ ./mem_usage.sh app.py
24MB
```

Sending *Some* Traffic:

```
/workspace/pycon $ ab -n50 http://127.0.0.1:5000/metrics/test_service
```

Memory Usage After:

```
~/workspace/pycon $ ./mem_usage.sh app.py
4070MB
```

There’s a leak!

Let’s Hunt!

Tools Available

Core dump analysis

Native libraries:

- objgraph
- memory-profiler
- Heapy
- tracemalloc
- ...

tracemalloc

because built-in, that's why!

tracemalloc

straight from the docs

- Traceback where an object was allocated
- Statistics on allocated memory blocks **per filename and per line number**.
- Compute the differences between two snapshots **to detect memory leaks**

tracemalloc

how to?

```
import tracemalloc


tracemalloc.start()

# ... do stuff ...

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
    print(stat)
```

Inside tracemalloc

and the magic it does!

Python Memory Operations API

- PyObject_Malloc/Free
- PyMem_Malloc/Free

Override them.

Trace the calls.

The App

again!

- Stateless
- Fetches metrics and serves them

The Master Plan

to catch the Leak

1. Warm up
    - by sending a couple requests
2. Snapshot
3. Send traffic
4. Snapshot again
5. Analysis

The Plan Executor

```
import tracemalloc
tracemalloc.start()
s=None

...
def get_service_metrics(service):
...

@app.route("/snapshot")
def snap():
    global s
    if not s:
        s = tracemalloc.take_snapshot()
        return "taken snapshot\n"
    else:
        lines = []
        top_stats = tracemalloc.take_snapshot().compare_to(s, 'lineno')
        for stat in top_stats[:5]:
            lines.append(str(stat))
        return "\n".join(lines)
```

Execution

Warm up:

```
~/workspace/pycon $ ab -n2 http://127.0.0.1:5000/metrics/test_service
~/workspace/pycon $ curl 127.0.0.1:5000/snapshot
taken snapshot
```

Sending *Some* Traffic:

```
/workspace/pycon $ ab -n50 http://127.0.0.1:5000/metrics/test_service
```

drumrolls

The leak:

```
~/workspace/pycon $ curl 127.0.0.1:5000/snapshot
workspace/pycon/helpers.py:25: size=4206 MiB (+4034 MiB), count=1 (+0), average=4206 MiB
/usr/local/lib/python3.7/site-packages/werkzeug/serving.py:226: size=21.7 KiB (+18.5 KiB),
count=20 (+17), average=1112 B
/usr/local/Cellar/python/3.7.4/Frameworks/Python.framework/Versions/3.7/lib/python3.7/socket
.py:253: size=14.4 KiB (+5168 B), count=42 (+34), average=350 B
/usr/local/lib/python3.7/site-packages/werkzeug/_compat.py:193: size=4128 B (+3534 B),
count=60 (+51), average=69 B
/usr/local/lib/python3.7/site-packages/werkzeug/serving.py:223: size=3640 B (+3094 B),
count=60 (+51), average=61 B
```

The Culprit

```
from flask import jsonify

PERF_DATA = dict()

def json_response(func):
    def wrapper(*args, **kwargs):
        global PERF_DATA
        start = time.time()
        r_val = func(*args, **kwargs)
        time_taken = start - time.time()

        # amplify the leak
        time_taken = [time_taken]*10000000
        # update performance data
        key = str((args, kwargs))
        if key in PERF_DATA:
            PERF_DATA[key].extend(time_taken)
        else:
            PERF_DATA[key] = time_take
        return jsonify(r_val)

    return wrapper
```

Recap

We

- Understood the memory management
- Understood the application behaviour
- Hunted the leak!

That’s all for today...

My home is @ https://sanket.plus