What's Eating My Memory?

Andrew Bennetts
andrew@bemusement.org
Bazaar team, Canonical
The problem
So you're worried about how much memory your program is using...
You take the usual steps...
1. Remove your `__del__` methods
2. Sprinkle `__slots__` everywhere
3. Avoid cycles
Ok `__del__` and `__slots__`, but your program still consumes 2GB of memory?
Anatomy of a PyObject
typedef struct _object {
    PyObject_VAR_HEAD
} PyObject;

typedef struct {
    PyObject_VAR_VAR_HEAD
} PyVarObject;
/* PyObject_HEAD defines the initial segment of every PyObject. */
#define PyObject_HEAD
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
```c
#define PyObject_VAR_HEAD               \ 
   PyObject_HEAD                       \ 
Py_ssize_t ob_size; /* Number of items in variable part */
```
What the heck does that mean?
Some Python object sizes:
(on 32-bit CPython 2.6)
<table>
<thead>
<tr>
<th>type</th>
<th>size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>12</td>
</tr>
<tr>
<td>str</td>
<td>24</td>
</tr>
<tr>
<td>tuple</td>
<td>24 + len(t) * 4</td>
</tr>
<tr>
<td>dict, len &lt; 4</td>
<td>132</td>
</tr>
<tr>
<td>dict, len &lt; 22</td>
<td>1672</td>
</tr>
<tr>
<td>instance of &quot;class C(object): pass&quot;</td>
<td>32</td>
</tr>
</tbody>
</table>
How can you find these numbers?
Answer 1:

```python
sys.getsizeof(thing)
```
Answer 2:

Just instantiate 100000 of something and see how much memory it takes!
An aside:

Memory stats on Linux
$ cat /proc/PID/status | grep Vm
VmPeak: 8484 kB
VmSize: 8484 kB
VmLck: 0 kB
VmHWM: 4372 kB
VmRSS: 4372 kB
VmData: 1680 kB
VmStk: 100 kB
VmExe: 1920 kB
VmLib: 3964 kB
VmPTE: 32 kB
Goodies built into Python
gc.get_referrers

gc.get_objects

sys.getsizeof

sys.getrefcount
These can give a good approximation of “which kinds of objects are most used?”, etc.
def mostRefs(n=30):
    d = {}
    for obj in gc.get_objects():
        if type(obj) is types.InstanceType:
            cls = obj.__class__
        else:
            cls = type(obj)
    d[cls] = d.get(cls, 0) + 1
    counts = [(x[1],x[0]) for x in d.items()]
    counts = sorted(counts)[-n:]
    return reversed(counts)

def printCounts(counts=None, file=None):
    if counts is None: counts = mostRefs()
    if file is None: file = sys.stdout
    for c, obj in counts:
        file.write(“%s %s
” % (c, obj))
Sadly, sys and gc modules have some blind spots

Besides, who wants to write this stuff from scratch?
Meliae
https://launchpad.net/meliae

Two parts: scanner and processor
scanner — dump memory details to disk

processor — analyse those dumps
Designed for use from Python prompt (or pdb):
> /home/andrew/code/bzr/bzrlib/inventory.py(1501)__init__()-->None
-> self.root_id = None
(Pdb) from meliae import scanner
(Pdb) pp self
<bzrlib.inventory.CHKInventory object at 0xb0ce1ac>
(Pdb) scanner.get_recursive_size(self)
(854591, 47393254)
(Pdb) scanner.dump_all_objects('filename.json')
(Pdb)
```python
>>> from meliae import loader
>>> om = loader.load('/tmp/bzr-branch.bzr-dev.meliae')
loaded line 903085, 903086 objs, 107.7 / 107.7 MiB read in 35.8s
>>> om.summarize()
Total 903086 objects, 249 types, Total size = 50.5MiB (52987209 bytes)

<table>
<thead>
<tr>
<th>Index</th>
<th>Count</th>
<th>%</th>
<th>Size</th>
<th>%</th>
<th>Cum</th>
<th>Max</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>221788</td>
<td>24</td>
<td>18699847</td>
<td>35</td>
<td>35</td>
<td>1878706</td>
<td>str</td>
</tr>
<tr>
<td>1</td>
<td>8790</td>
<td>0</td>
<td>13457328</td>
<td>25</td>
<td>60</td>
<td>1573000</td>
<td>dict</td>
</tr>
<tr>
<td>2</td>
<td>226283</td>
<td>25</td>
<td>8097956</td>
<td>15</td>
<td>75</td>
<td>276</td>
<td>tuple</td>
</tr>
<tr>
<td>3</td>
<td>292078</td>
<td>32</td>
<td>3504936</td>
<td>6</td>
<td>82</td>
<td>12</td>
<td>bzrlib._static_tuple_c.StaticTuple</td>
</tr>
<tr>
<td>4</td>
<td>213</td>
<td>0</td>
<td>2926384</td>
<td>5</td>
<td>88</td>
<td>1048688</td>
<td>set</td>
</tr>
<tr>
<td>5</td>
<td>1019</td>
<td>0</td>
<td>2631000</td>
<td>4</td>
<td>93</td>
<td>1339048</td>
<td>list</td>
</tr>
<tr>
<td>6</td>
<td>120034</td>
<td>13</td>
<td>1440408</td>
<td>2</td>
<td>95</td>
<td>12</td>
<td>int</td>
</tr>
<tr>
<td>7</td>
<td>1476</td>
<td>0</td>
<td>661248</td>
<td>1</td>
<td>97</td>
<td>448</td>
<td>type</td>
</tr>
<tr>
<td>8</td>
<td>8268</td>
<td>0</td>
<td>562224</td>
<td>1</td>
<td>98</td>
<td>68</td>
<td>code</td>
</tr>
<tr>
<td>9</td>
<td>8194</td>
<td>0</td>
<td>458864</td>
<td>0</td>
<td>98</td>
<td>56</td>
<td>function</td>
</tr>
<tr>
<td>10</td>
<td>4988</td>
<td>0</td>
<td>139664</td>
<td>0</td>
<td>99</td>
<td>28</td>
<td>_LazyGroupCompressFactory</td>
</tr>
<tr>
<td>11</td>
<td>1723</td>
<td>0</td>
<td>68920</td>
<td>0</td>
<td>99</td>
<td>40</td>
<td>weakref</td>
</tr>
<tr>
<td>12</td>
<td>1163</td>
<td>0</td>
<td>41868</td>
<td>0</td>
<td>99</td>
<td>36</td>
<td>wrapper_descriptor</td>
</tr>
<tr>
<td>13</td>
<td>1171</td>
<td>0</td>
<td>37472</td>
<td>0</td>
<td>99</td>
<td>32</td>
<td>builtin_function_or_method</td>
</tr>
<tr>
<td>14</td>
<td>790</td>
<td>0</td>
<td>25280</td>
<td>0</td>
<td>99</td>
<td>32</td>
<td>getset_descriptor</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>0</td>
<td>18240</td>
<td>0</td>
<td>99</td>
<td>592</td>
<td>StgDict</td>
</tr>
<tr>
<td>16</td>
<td>42</td>
<td>0</td>
<td>16444</td>
<td>0</td>
<td>99</td>
<td>544</td>
<td>frame</td>
</tr>
<tr>
<td>17</td>
<td>494</td>
<td>0</td>
<td>15808</td>
<td>0</td>
<td>99</td>
<td>32</td>
<td>method_descriptor</td>
</tr>
<tr>
<td>18</td>
<td>299</td>
<td>0</td>
<td>13156</td>
<td>0</td>
<td>99</td>
<td>44</td>
<td>_LRUNode</td>
</tr>
<tr>
<td>19</td>
<td>203</td>
<td>0</td>
<td>12532</td>
<td>0</td>
<td>99</td>
<td>348</td>
<td>unicode</td>
</tr>
</tbody>
</table>
```
```python
>>> om[s.summaries[0].max_address]
str(3068231688 1878706B
'f\x92\x02chkinventory:\nsearch_key_name: hash-255-
way\nroot_id: TREE_ROOT\nparent_id_basename_to_file_id:
sha')
```
```
>>> om[s.summaries[1].max_address]
dict(172895468 1573000B 63308refs)
```
```
>>> om[s.summaries[1].max_address].p
[]
```
```
>>> om[s.summaries[1].max_address].c[:3]
[bzrlib._static_tuple_c.StaticTuple(181272096 12B 1refs),
bzrlib._static_tuple_c.StaticTuple(210597672 12B 2refs),
bzrlib._static_tuple_c.StaticTuple(174057584 12B 1refs)]
```

[Address, size, children, parents, content]
heapy
http://guppy-pe.sourceforge.net/#Heapy

Powerful... but poorly documented
```python
from guppy import hpy
hp = hpy()
h = hp.heap()

Partition of a set of 1449133 objects. Total size = 102766644 bytes.

<table>
<thead>
<tr>
<th>Index</th>
<th>Count</th>
<th>%</th>
<th>Size</th>
<th>%</th>
<th>Cumulative</th>
<th>%</th>
<th>Kind (class / dict of class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>985931</td>
<td>68</td>
<td>46300932</td>
<td>45</td>
<td>46300932</td>
<td>45</td>
<td>str</td>
</tr>
<tr>
<td>1</td>
<td>24681</td>
<td>2</td>
<td>22311624</td>
<td>22</td>
<td>68612556</td>
<td>67</td>
<td>dict of pkgcore.ebuild.ebuild_src.package</td>
</tr>
<tr>
<td>2</td>
<td>49391</td>
<td>3</td>
<td>21311864</td>
<td>21</td>
<td>89924420</td>
<td>88</td>
<td>dict (no owner)</td>
</tr>
<tr>
<td>3</td>
<td>115974</td>
<td>8</td>
<td>3776948</td>
<td>4</td>
<td>93701368</td>
<td>91</td>
<td>tuple</td>
</tr>
<tr>
<td>4</td>
<td>152181</td>
<td>11</td>
<td>3043616</td>
<td>3</td>
<td>96744984</td>
<td>94</td>
<td>long</td>
</tr>
<tr>
<td>5</td>
<td>36009</td>
<td>2</td>
<td>1584396</td>
<td>2</td>
<td>98329380</td>
<td>96</td>
<td>weakref.KeyedRef</td>
</tr>
<tr>
<td>6</td>
<td>11328</td>
<td>1</td>
<td>1540608</td>
<td>1</td>
<td>99869988</td>
<td>97</td>
<td>dict of pkgcore.ebuild.ebuild_src.ThrowAwayNameSpace</td>
</tr>
<tr>
<td>7</td>
<td>24702</td>
<td>2</td>
<td>889272</td>
<td>1</td>
<td>100759260</td>
<td>98</td>
<td>types.MethodType</td>
</tr>
<tr>
<td>8</td>
<td>11424</td>
<td>1</td>
<td>851840</td>
<td>1</td>
<td>101611100</td>
<td>99</td>
<td>list</td>
</tr>
<tr>
<td>9</td>
<td>24681</td>
<td>2</td>
<td>691068</td>
<td>1</td>
<td>102302168</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

pkgcore.ebuild.ebuild_src.package
<54 more rows. Type e.g. '_.more' to view.>
```
>>> h.get_rp(40)
Reference Pattern by <[dict of] class>.
  0: _ --- [-] 14 (dict (no owner) | list | str | types.FrameType | types.Gen...
  1: a      [-] 3 dict (no owner): 0x8c11f34*2, 0x8c1bd54*2, 0x8c1f854*2
  2: aa ---- [-] 1 list: 0x833c504*18
  3: a3     [-] 1 dict of django.db.backends.mysql.base.DatabaseWrapper: 0x...
  4: a4 ------ [-] 1 dict (no owner): 0x83a65d4*2
  5: a5      [R] 1 guppy.heapy.heapyc.RootStateType: 0xb787c7a8L
  6: a3b ---- [-] 1 django.db.backends.mysql.base.DatabaseWrapper: 0x8356a34
  7: a3ba    [S] 7 dict of module: ..db, ..models, ..query, ..transaction...
  8: b ---- [S] 1 types.FrameType: <<lambda> at 0x8b16ecc>
  9: c      [-] 2 list: 0x833c504*18, 0xb7dafe6cL*5
<Type e.g. '_.more' for more.>
Also has support for taking differences between heaps, sorting the data, grouping the data...
...and probably much more, if you can understand the manual.
Has had portability problems :(
objgraph
(tired of text yet?)
http://mg.pov.lt/objgraph/

>>> import objgraph
>>> objgraph.show_refs([obj])
>>> import objgraph
>>> x = []
>>> y = [x, [x], dict(x=x)]
>>> objgraph.show_refs([y], filename='sample.png')
Graph written to /tmp/tmp3onTjF.dot (5 nodes)
Image generated as sample.png
Can also graph backrefs, define filters, highlights, and more.
valgrind
Try this if you suspect a C extension is leaking
For best results, compile your own Python following the instructions in Misc/README.valgrind
Typical use:

valgrind python foo.py
Thanks!

Links, slides — http://bemusement.org/pycon2010

(soon, I promise!)