Richard WM Jones

AUGUST 7, 2009 · 4:10 PM

OCaml internals part 4: The major heap

[What's this all about? Start at part 1]

In part 3 we looked at the minor heap. In this part we'll look at the other bit, not surprisingly called the major heap.

The **major heap** is the main OCaml memory store. It is composed of some number of large chunks of memory (allocated by C malloc).

When the minor heap runs out, it triggers a minor collection.

The minor collection starts at all the local roots and "oldifies" them, basically copies them by reallocting those objects (recursively) to the major heap. After this, any objects left in the minor heap are unreachable, so the minor heap can be reused just by resetting `caml_young_ptr`.

```
           reachable        reachable
+---------+---------+------+----+----+----------------+
|         |/////////|      |///-->///|                |
+---------+---------+------+----+----+----------------+
           ^                ^
           |                |
        local root       local root
```

Most objects in a typical program are short-lived and never make it out of the minor heap. At least, that is the assumption, and allocation in functional programs is only cheap if this assumption holds. A few programs won't work like this at all, and some programs may require adjustment of the minor heap using tunables in the OCaml `Gc` module.

One thing you should notice from the discussion of values and blocks in parts 1 and 2 is that at runtime the garbage collector always knows what is a pointer, and what is an int or opaque data (like a string). Pointers get scanned so the GC can find unreachable blocks. Ints and opaque data must *not* be scanned. This is the reason for having a tag bit for integer-like values, and one of the uses of the tag byte in the header.

```
                  "Tag byte space"
+----------+
| 0        | Array, tuple, etc.
```

```
+----------+
| 1        |
+----------+
| 2        |
~          ~
|          | Tags in the range 0..245 are used for variants
~          ~
| 245      |
+----------+
| 246      | Lazy (before being forced)
+----------+
| 247      | Closure
+----------+
| 248      | Object                        ^
+----------+                               |  Block contains
| 249      | Used to implement closures    |  values which the
+----------+                               |  GC should scan
| 250      | Used to implement lazy values |
+----------+ <--------------------------  No_scan_tag
| 251      | Abstract data                 |
+----------+                               |  Block contains
| 252      | String                        |  opaque data
+----------+                               |  which GC must
| 253      | Double                        V  not scan
+----------+
| 254      | Array of doubles
+----------+
| 255      | Custom block
+----------+
```

So in the normal course of events, a small, long-lived object will start on the minor heap and be copied once into the major heap. Large objects go straight to the major heap.

The major heap as I said is composed of large chunks of memory allocated through C malloc. Inside these chunks there is quite a complex allocation scheme used, which I won’t go into. But there is another important structure used in the major heap, called the **page table**. The garbage collector must at all times know which pieces of memory belong to the major heap, and which pieces of memory do not, and it uses the page table to track this.

One reason why we always want to know where the major heap lies is so we can avoid scanning pointers which point to C structs outside the OCaml heap. The GC will not stray beyond its own heap, and treats all pointers outside as opaque (it doesn’t touch them or follow them).

In OCaml ≤ 3.10 the page table was implemented as a simple bitmap, with 1 bit per page of virtual memory (major heap chunks are always page-aligned). This was unsustainable for 64 bit address spaces where memory allocations can be *very very* far apart, so in OCaml 3.11 this was changed to a sparse hash table.

```
bitmap      pages of main memory
+-----+     +-------------------+
| 1   |     | chunk of major    |
+-----+     + heap              +
| 1   |     |                   |
+-----+     +-------------------+
| 0   |       other memory
+-----+     +-------------------+
| 1   |     | chunk of major    |
+-----+     + heap              +
| 1   |     |                   |
+-----+     +                   +
| 1   |     |                   |
+-----+     +-------------------+
```

Because of the page table, C pointers can be stored directly as values, which saves time and space. (However if your C pointer later gets freed, you must NULL out the value — the reason is that the same memory address might later get malloc'd for the OCaml major heap, thus suddenly becoming a "valid" address again. This usually results in a crash).

In part 5 I'll look at the workings of the garbage collector in a little more detail.

Twitter

Email

Print

One Response to *OCaml internals part 4: The major heap*

Pingback: A beginners guide to OCaml internals « Richard WM Jones