Coping with, and minimizing, memory usage in Common Lisp (SBCL)

Asked 15 years, 8 months ago Modified 15 years, 7 months ago

Viewed 5k times

14

I have a VPS with not very much memory (256Mb) which I am trying to use for Common Lisp development with SBCL+Hunchentoot to write some simple web-apps. A large amount of memory appears to be getting used without doing anything particularly complex, and after a while of serving pages it runs out of memory and either goes crazy using all swap or (if there is no swap) just dies.

So I need help to:

- Find out what is using all the memory (if it's libraries or me, especially)
- Limit the amount of memory which SBCL is allowed to use, to avoid massive quantities of swapping
- Handle things cleanly when memory runs out, rather than crashing (since it's a web-app I want it to carry on and try to clean up).

I assume the first two are reasonably straightforward, but is the third even possible? How do people handle out-of-

memory or constrained memory conditions in Lisp?

(Also, I note that a 64-bit SBCL appears to use literally twice as much memory as 32-bit. Is this expected? I can run a 32-bit version if it will save a lot of memory)

memory lisp common-lisp out-of-memory sbcl

Share

Improve this question

Follow

edited May 15, 2009 at 10:57

Pablo Fernandez

287k ● 139 ● 401 ● 641

asked Apr 2, 2009 at 8:52

David Gardner

7,843 ● 5 ● 37 ● 37

4 Answers

Sorted by: Highest score (default)

16

To limit the memory usage of SBCL, use --dynamic-space-size option (e.g., `sbcl --dynamic-space-size 128` will limit memory usage to 128M).

To find out who is using memory, you may call `(room)` (the function that tells how much memory is being used) at different times: at startup, after all libraries are loaded and then during work (of cource, call `(sb-ext:gc :full t)` before room not to measure the garbage that has not yet been collected).

Also, it is possible to use SBCL Profiler to measure memory allocation.

Share Improve this answer Follow

edited Apr 2, 2009 at 21:30

answered Apr 2, 2009 at 9:53

dmitry_vk
4,459 ● 21 ● 23

Could you add a link to something about the SBCL profiler, please? :) – David Gardner Apr 3, 2009 at 16:10

2 There is a description of profiler in the SBCL manual at sbcl.org/manual/Deterministic-Profiler.html – dmitry_vk Apr 4, 2009 at 14:59

5

> Find out what is using all the memory (if it's libraries or me, especially)

Attila Lendvai has some SBCL-specific code to find out where an allocated objects comes from. Refer to http://article.gmane.org/gmane.lisp.steel-bank.devel/12903 and write him a private mail if needed.

Be sure to try another implementation, preferably with a precise GC (like Clozure CL) to ensure it's not an implementation-specific leak.

> Limit the amount of memory which SBCL is allowed to use, to avoid massive quantities of swapping

Already answered by others.

> Handle things cleanly when memory runs out, rather than crashing (since it's a web-app I want it to carry on and try to clean up).

256MB is tight, but anyway: schedule a recurring (maybe 1s) timed thread that checks the remaining free space. If the free space is less than X then use exec() to replace the current SBCL process image with a new one.

Share Improve this answer

Follow

answered Apr 3, 2009 at 15:31

Leslie P. Polzer

3,028 ● 22 ● 19

3

If you don't have any type declarations, I would expect 64-bit Lisp to take twice the space of a 32-bit one. Even a plain (small) int will use a 64-bit chunk of memory. I don't think it'll use less than a machine word, unless you declare it.

I can't help with #2 and #3, but if you figure out #1, I suspect it won't be a problem. I've seen SBCL/Hunchentoot instances running for ages. If I'm

using an outrageous amount of memory, it's usually my own fault. :-)

Share Improve this answer

Follow

answered Apr 2, 2009 at 20:21

Ken

1,537 ● 1 ● 10 ● 13

1

I would not be surprised by a 64-bit SBCL using twice the meory, as it will probably use a 64-bit cell rather than a 32-bit one, but couldn't say for sure without actually checking.

Typical things that keep memory hanging around for longer than expected are no-longer-useful references that still have a path to the root allocation set (hash tables are, I find, a good way of letting these things linger). You could try interspersing explicit calls to GC in your code and make sure to (as far as possible) not store things in global variables.

Share Improve this answer

Follow

answered Apr 2, 2009 at 9:47

Vatine

21.2k ● 4 ● 56 ● 70