Ruby master - Bug #18952
rb_aligned_free: munmap failed
08/01/2022 11:31 PM - mdomsch-sz (Matt Domsch)

Status: Third Party's Issue
Priority: Normal
Assignee: peterzhu2118 (Peter Zhu)
Target version: ruby -v:
  ruby 3.1.2p20 (2022-04-12 revision 4491bb740a) [x86_64-linux-musl]

Backport:
  2.7: UNKNOWN, 3.0: UNKNOWN, 3.1: UNKNOWN

Description
Over the past several weeks, our Linux ruby 3.1.2 application has failed repeatedly with:

[BUG] rb_aligned_free: munmap failed

Ruby tracebacks indicate failures in many libraries and functions, due to the garbage collector running at its discretion. Reproduction is not easily accomplished as this is a low-level ruby function being invoked from within a large web application, and not directly by our application, the servers of which have 24GB RAM and are not seeing any indication of out-of-memory concerns.

Unfortunately the munmap() call here does not catch the return value and errno to report why munmap() failed.

I append one example backtrace log.

History
#1 - 08/02/2022 03:54 AM - mdomsch-sz (Matt Domsch)
Over the same 2-week period, I've seen two other logs that appear to be related:
[BUG] rb_aligned_malloc: munmap failed for end

#2 - 08/02/2022 12:42 PM - mdomsch-sz (Matt Domsch)
I note that the process has 65531 regions in the process memory map, which is basically identical to the default runtime limit vm.max_map_count = 65530. Our runtime environment is AWS Fargate, so it's unlikely we can raise this limit. Other reporters hitting this limit have likewise determined it can't be changed in Fargate.

#3 - 08/03/2022 02:36 AM - mame (Yusuke Endoh)
- Assignee set to peterzhu2118 (Peter Zhu)

@peterzhu2118 (Peter Zhu) Can you check it out?

This is just my guess but I think it has something to do with musl's strategy on how to use mmap/munmap. Are you using Alpine Linux? I wonder if it won't happen if you use Ubuntu with glibc.

#4 - 08/03/2022 03:35 AM - mdomsch-sz (Matt Domsch)
We are using the official Ruby Docker image ruby:3.1.2-alpine3.15 which provides musl. We came to the same conclusion today to try ruby:3.1.2-slim-bullseye which comes with glibc, and using the environment variable MALLOC_ARENA_MAX=2 (it would otherwise default to 32 in our Fargate configuration). We will try this out on 8/3/22 and report back, though it takes a few days to experience the memory fragmentation that leads to such high map count thus far. We have not taken the next step of using jemalloc, and will likely do so as a further test later in the week.

#5 - 08/05/2022 12:58 PM - mdomsch-sz (Matt Domsch)
We were able to get our application with ruby:3.1.2-slim-bullseye and MALLOC_ARENA_MAX=2 into production last night, and there were no observed failures since. Without the change we would have expected several hundred failures. This appears to be an appropriate solution for us. The number of memory maps per process is between 600 and 3200, well below the 65530 limit we were reaching frequently before.

#6 - 08/09/2022 04:52 PM - jeremyevans0 (Jeremy Evans)
- Status changed from Open to Third Party's Issue

Files
logs.gz 414 KB 08/01/2022 mdomsch-sz (Matt Domsch)