serverfault

How does vm.overcommit_memory work?

Asked 8 years, 6 months ago Modified 25 days ago Viewed 138k times

73

When I use the default settings:

```
vm.overcommit_memory = 0
vm.overcommit_ratio = 50
```

I can read these values from `/proc/meminfo` file:

```
CommitLimit:     2609604 kB
Committed_AS:    1579976 kB
```

But when I change `vm.overcommit_memory` from `0` to `2`, I'm unable to start the same set of applications that I could start before the change, especially amarok. I had to change `vm.overcommit_ratio` to `300`, so the limit could be increased. Now when I start amarok, `/proc/meminfo` shows the following:

```
CommitLimit:     5171884 kB
Committed_AS:    3929668 kB
```

This machine has only 1GiB of RAM, but amarok works without problems when `vm.overcommit_memory` is set to 0. But in the case of setting it to `2`, amarok needs to allocate over 2GiB of memory. Is it a normal behavior? If so, could anyone explain why, for instance, firefox (which consumes 4-6x more memory than amarok) works in the same way before and after the change?

memory kernel sysctl

Share Improve this question Follow

edited Jun 18, 2014 at 17:36

asked Jun 18, 2014 at 17:27

Mikhail Morfikov
916 1 10 12

What's the said `amarok` . – John Sep 9 at 7:39

2 Answers

Sorted by: Highest score (default)

97

You can find the documentation in `man 5 proc` (or at kernel.org):

```
    /proc/sys/vm/overcommit_memory
           This file contains the kernel virtual memory accounting mode.
           Values are:

                  0: heuristic overcommit (this is the default)
                  1: always overcommit, never check
                  2: always check, never overcommit

           In mode 0, calls of mmap(2) with MAP_NORESERVE are not
           checked, and the default check is very weak, leading to the
           risk of getting a process "OOM-killed".

           In mode 2 (available since Linux 2.6), the total virtual
           address space that can be allocated (CommitLimit in /proc/mem-
           info) is calculated as

               CommitLimit = (total_RAM - total_huge_TLB) *
                             overcommit_ratio / 100 + total_swap
```

The simple answer is that setting overcommit to 1, will set the stage so that when a program calls something like `malloc()` to allocate a chunk of memory (`man 3 malloc`), it will always succeed regardless if the system knows it will not have all the memory that is being asked for.

The underlying concept to understand is the idea of *virtual memory*. Programs see a virtual address space that may, or may not, be mapped to actual physical memory. By disabling overcommit checking, you tell the OS to just assume that there is always enough physical memory to backup the virtual space.

Example

To highlight why this can sometimes matter, take a look at the Redis guidances on why `vm.overcommit_memory` should be set to 1 for it.

Share Improve this answer Follow

edited Nov 21 at 12:35
Andrew Vasylchuk
103 3

answered Jun 18, 2014 at 17:49
Kyle Brandt
82.6k 71 302 444

2 But shouldn't the value of `Committed_AS` be the same in both cases? – Mikhail Morfikov Jun 18, 2014 at 18:05

@MikhailMorfikov: In theory, I believe so, but who knows what these programs are doing. Would want to see a more controlled environment with a simple program that just allocates say a gig of ram via Malloc. And then run the test after rebooting between tests. – Kyle Brandt Jun 18, 2014 at 19:18

Ok, so I will stay with `0` for now. – Mikhail Morfikov Jun 18, 2014 at 19:21

2 @MikhailMorfikov: Yes, practically I think 0 makes the most sense. In my environment, the only time I enable 1 is for Redis, which does stuff where it *expects* do be asking for a lot more memory that it is using due to a fork(). The child will pretty much use all the same memory pages, but Linux doesn't know that says to be safe it has to assume 2x memory will be used (if you want to learn more: redis.io/topics/faq) – Kyle Brandt Jun 18, 2014 at 19:25

shouldn't the last statement in your answer begin like "by enabling overcommit"? because setting it to 1 means you're asking it to overcommit, right? – asgs Oct 24, 2018 at 20:00

21

This is an old question with a well-established answer, but I think there's more to add.

First of all, when `vm.overcommit_memory = 0`, the `vm.overcommit_ratio` value is irrelevant. The kernel will use a heuristic algorithm to overcommit memory, so that your `amarok` process can be allocated more memory than is available.

When you set `vm.overcommit_memory` to `2`, the `vm.overcommit_ratio` value becomes relevant. By default, this value is set to `50`, which means the system would only allocate up to 50% of your RAM (plus swap). This explains why you are unable to start programs that was fine when `vm.overcommit_memory = 0` - because there's less than 500MB of allocatable memory (assuming no swap).

When you set it to `300`, you are allowing the system to allocate up to 300% of your RAM (plus swap, if any), which is why the `CommitLimit` value in `/proc/meminfo` is so high.

Although `vm.overcommit_memory = 2` is usually used to prevent overcommitment, here, you're using it to cap the amount that can be overcommitted. Setting it to `300` is dangerous as your system don't have `5171884 kB` of memory, and so, depending on how much swap space you have, the system will use swap (which is slow), or will run out of memory altogether.

As to why `amarok` uses more memory when `vm.overcommit_memory = 2` - this is probably because `amarok` works best with more memory, but is fine with less as well. So the logic of the program may try to allocate 2GB of memory initially, but if it fails, try 1GB.

Share Improve this answer Follow

answered Dec 5, 2019 at 11:43
d4nyll
334 2 9