



Analyzing Memory Usage in Red Hat Enterprise Linux

Author Name: Allison Pranger
Technical Reviewer: Casey Dahlin
Editor: Chris Negus
09/26/11

OVERVIEW

Lack of free memory can either be a symptom of a bigger problem or nothing to worry about at all. Different operating systems handle memory in different ways, so it is important to start with a basic understanding of how memory is handled by the Linux kernel.

Most questions related to memory usage in Red Hat Enterprise Linux fall into one of a few categories:

- Why don't I have more free memory?
- How much memory is this process taking up?
- How much memory is the kernel taking up?

This tech brief provides answers to these questions and will give you a basic overview of how to analyze memory usage in Red Hat Enterprise Linux so that you can more easily pinpoint possible issues.

WHY DON'T I HAVE MORE FREE MEMORY?

While examining memory statistics, you might encounter an unexpected value, such as if only 4% of your memory is identified as free. This is not necessarily a cause for concern, even if you are not running a lot of processes. The Linux kernel is coded off a philosophy that an unused resource is a wasted resource, and it has ways of using available resources regardless of your situation.

The example below lists the contents of `/proc/meminfo` for a Red Hat Enterprise Linux system with roughly 20% listed free memory:

```
MemTotal:      509856 kB
MemFree:       109400 kB  <-- amount of memory listed as free
Buffers:       8524 kB
Cached:        149840 kB  <-- amount cached
SwapCached:    88 kB
Active:        198704 kB
Inactive:      41208 kB
HighTotal:     0 kB
HighFree:     0 kB
LowTotal:     509856 kB
LowFree:      109400 kB
SwapTotal:    1048568 kB
SwapFree:     1048476 kB
Dirty:         28 kB      <-- amount of dirty cache
Writeback:     0 kB
AnonPages:    81488 kB
Mapped:       13860 kB
Slab:         41928 kB
PageTables:   4736 kB
NFS_Unstable: 0 kB
Bounce:       0 kB
```



```
CommitLimit:      1252296 kB
Committed_AS:     262004 kB
VmallocTotal:     34359738367 kB
VmallocUsed:      1212 kB
VmallocChunk:     34359736039 kB
HugePages_Total:  50
HugePages_Free:   49
HugePages_Rsvd:   8
Hugepagesize:    2048 kB
```

The Linux kernel attempts to optimize I/O performance by copying what is on the disk into memory for faster access. The amount of memory used by the cache is listed in `/proc/meminfo` (noted above). Cached memory can be freed quickly if memory is needed for other reasons. However, there are two types of cached pages, and the amount of cached memory that can be evicted depends on how much of the cache is considered dirty.

Dirty cached pages are those that contain changes that the system still needs to write to disk. Whenever the system needs to reclaim memory, it can evict clean cached pages, but dirty cached pages must first be copied back to disk.

Processing can become less efficient if you have a lot of write-heavy operations and your dirty cache is large. There are several tunables you can adjust to reduce the amount of data cached by the Linux kernel. The most effective in this case is `dirty_background_ratio`, which contains, as a percentage of total system memory, the number of pages at which the `pdflush` background writeback daemon will start writing out dirty data. The default value is 10 percent. To limit the size of the page cache, decrease the value so the `pdflush` daemon will start writing out dirty data sooner. For example, to temporarily change the size limit to 8 percent, type the following command:

```
# sysctl -w vm.dirty_background_ratio=8
```

To make the change permanent, you must add the following line to the `/etc/sysctl.conf` file, where `#` is the desired value:

```
vm.dirty_background_ratio = 8
```

The new `vm.dirty_background_ratio` value will take effect on your next reboot (or type `sysctl -p` to reprocess the `/etc/sysctl.conf` file immediately).

LEARN MORE: To learn more about how to control the size of the page cache, see the Red Hat Knowledgebase article at <https://access.redhat.com/kb/docs/DOC-41749>.

HOW MUCH MEMORY IS THIS PROCESS TAKING UP?

The figure below illustrates a process's address space from the process's point of view. There are both mapped and unmapped areas. Mapped areas return information, but if the process attempts to access an unmapped area, it will encounter a segmentation fault.

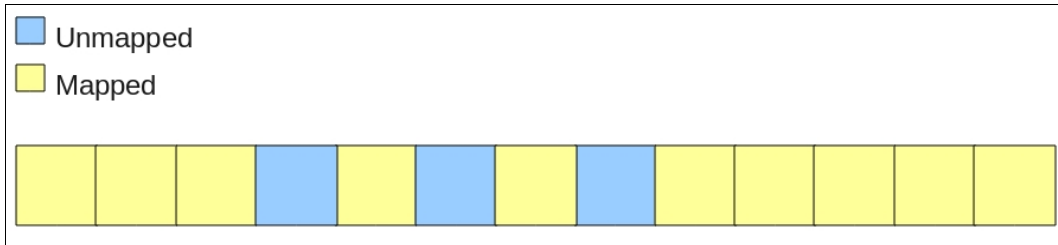


Figure 1: Process Address Space

You might be tempted to assume that the mapped areas relate directly to the memory used by the process. In this example, that would translate to about 40k of memory (10 pages at about 4k each). However, that is not the whole story. As Figure 2 illustrates, a lot more is happening behind the scenes.

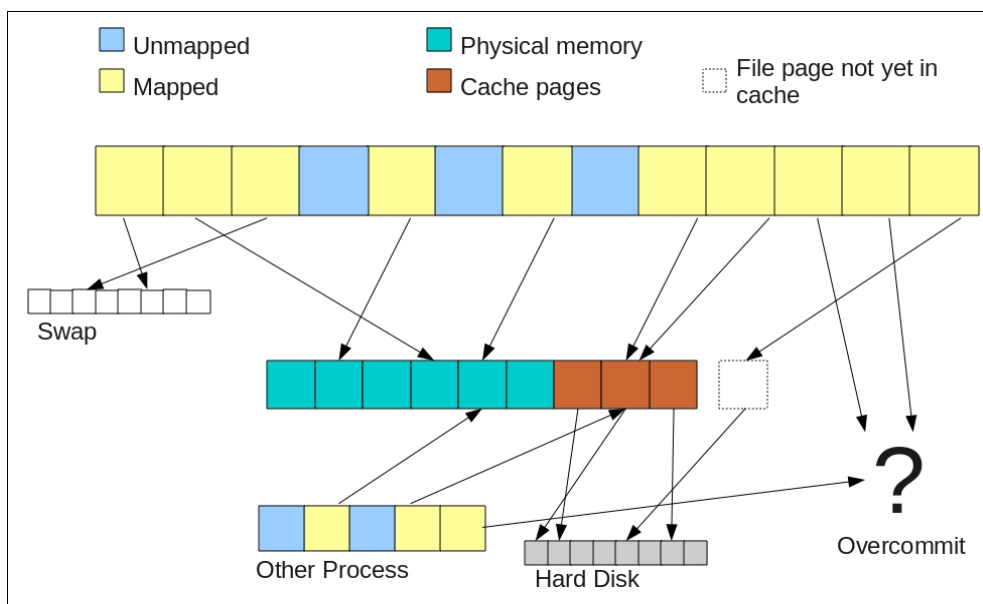


Figure 2: Actual Process Memory Usage

As you can see in the example in Figure 2, two pages of memory are saved to disk on a swap partition, one is an empty file page not yet cached, and two are pages allocated to memory that have not yet been used (overcommit). The actual number of pages used by the process is four (three physical-memory pages and one cached page). This is a more accurate view of memory usage for the process.

At the bottom of Figure 2 is another process that is using some of the same pages. If you summed the VmRSS values from the two processes, you would get a total of six, even though the actual number of pages used is four (two pages are shared between the processes). If you killed the process at the top, you would only free two pages because the other process is holding onto the cache page and one of the physical-memory pages.

LEARN MORE: To see memory use by process and to watch it change with commands like **top** and **vmstat**, see the tech brief *Troubleshooting Performance Problems from Within the System* at



<https://access.redhat.com/knowledge/techbriefs/investigating-performance-issues-red-hat-enterprise-linux-part-3-4-troubleshoot>

HOW MUCH MEMORY IS THE KERNEL TAKING UP?

The memory used by the Linux kernel can be found by adding three of the `/proc/meminfo` values: **slab**, **dirty**, and **buffers**. Take another look again at the example given earlier:

```
MemTotal:    509856 kB
MemFree:     109400 kB
Buffers:     8524 kB      <-- data read off disk
Cached:      149840 kB
SwapCached:  88 kB
Active:      198704 kB
Inactive:    41208 kB
HighTotal:   0 kB
HighFree:    0 kB
LowTotal:    509856 kB
LowFree:     109400 kB
SwapTotal:   1048568 kB
SwapFree:    1048476 kB
Dirty:       28 kB      <-- amount of dirty cache
Writeback:   0 kB
AnonPages:   81488 kB
Mapped:      13860 kB
Slab:        41928 kB   <-- kernel memory
PageTables:  4736 kB
NFS_Unstable: 0 kB
Bounce:      0 kB
CommitLimit: 1252296 kB
Committed_AS: 262004 kB
VmallocTotal: 34359738367 kB
VmallocUsed:  1212 kB
VmallocChunk: 34359736039 kB
HugePages_Total: 50
HugePages_Free: 49
HugePages_Rsvd: 8
Hugepagesize: 2048 kB
```

The **buffers** value reflects the amount of data read off the disk that is not part of a file, which includes data structure information that points to actual files (for example, ext4 inode information).

Most of the memory used by the Linux kernel is listed under **slab**. When the kernel allocates memory out of the slab cache, it is labeled, and the purpose for which the kernel allocates the memory is recorded in `/proc/slabinfo`.

Below is an example of content found in `/proc/slabinfo`. Most memory is well labeled. However, there is a layer here that is similar to **malloc** in user space, reflected in the final three items:

```
$ cat /proc/slabinfo
slabinfo - version: 2.1
# name <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables
<limit> <batchcount> <sharedfactor> : slabdata <active_slabs> <num_slabs>
<sharedavail>
```



```
nfs_inode_cache 94 124 1056 31 8 : tunables 0 0 0 : slabdata 4 4 0
fscache_cookie_jar 102 102 80 51 1 : tunables 0 0 0 : slabdata 2 2 0
fuse_request 0 0 632 25 4 : tunables 0 0 0 : slabdata 0 0 0
fuse_inode 0 0 768 21 4 : tunables 0 0 0 : slabdata 0 0 0
...
kmalloc-256 2076 2240 256 32 2 : tunables 0 0 0 : slabdata 70 70 0
kmalloc-128 2096 2688 128 32 1 : tunables 0 0 0 : slabdata 84 84 0
kmalloc-64 17769 26368 64 64 1 : tunables 0 0 0 : slabdata 412 412 0
```

When you use the `/proc/slabinfo` interface to access slab cache and allocate memory in the kernel, the memory is labeled in a numbered bucket at the bottom of `slabinfo`. Once there, it can be difficult to tell precisely where the memory went.

If you have the `crash` program, you can start to trace `kmalloc` segments to find meaningful labels. Begin by searching through kernel memory using `search -k` within `crash` to get a list of all the places that an address is referenced. You can then use `kmem -S` within `crash`, which lists the addresses of every individual object that was allocated out of the slab cache, to correlate the address to other objects. This information will help you pinpoint which process is eating the most memory so that you can focus your attention on fixing any resulting issues.

Why Is the Kernel Using Swap When There Are Cached Pages Available?

When the kernel swaps out a page, that page stays swapped out until it is used again. This means that if there is a sudden spike in memory usage and pages get swapped out, those pages will not be swapped back in as soon as memory becomes free. The pages will not return until the application to which the pages belong tries to access them.

It should also be noted that Linux tends to favor clearing out the pages used least frequently, regardless of whether they are cached pages that need to be cleared or normal pages that need to be swapped. It might make more sense according to the kernel's heuristics to swap out a page rather than to free the cache. There is some bias in which action the kernel will prefer, though, and how much is tunable by `/proc/sys/vm/swappiness`.