

Overview of memory management

Traditional Unix tools like 'top' often report a surprisingly small amount of free memory after a system has been running for a while. For instance, after about 3 hours of uptime, the machine I'm writing this on reports under 60 MB of free memory, even though I have 512 MB of RAM on the system. Where does it all go?

The biggest place it's being used is in the disk cache, which is currently over 290 MB. This is reported by top as "cached". Cached memory is essentially free, in that it can be replaced quickly if a running (or newly starting) program needs the memory.

The reason Linux uses so much memory for disk cache is because the RAM is wasted if it isn't used. Keeping the cache means that if something needs the same data again, there's a good chance it will still be in the cache in memory. Fetching the information from there is around 1,000 times quicker than getting it from the hard disk. If it's not found in the cache, the hard disk needs to be read anyway, but in that case nothing has been lost in time.

To see a better estimation of how much memory is really free for applications to use, run the command:

```
$ free -m
```

The -m option stands for megabytes, and the output will look something like this:

	total	used	free	shared	buffers	cached
Mem:	503	451	52	0	14	293
-/+ buffers/cache:	143	360				
Swap:	1027	0	1027			

The -/+ buffers/cache line shows how much memory is used and free from the perspective of the applications. Generally speaking, if little swap is being used, memory usage isn't impacting performance at all.

Notice that I have 512 MB of memory in my machine, but only 503 is listed as available by free. This is mainly because the kernel can't be swapped out, so the memory it occupies could never be freed.

There may also be regions of memory reserved for/by the hardware for other purposes as well, depending on the system architecture.

The mysterious 880 MB limit on x86

By default, the Linux kernel runs in and manages only low memory. This makes managing the page tables slightly easier, which in turn makes memory accesses slightly faster. The downside is that it can't use all of the memory once the amount of total RAM reaches the neighborhood of 880 MB. This has historically not been a problem, especially for desktop machines.

To be able to use all the RAM on a 1GB machine or better, the kernel needs recompiled. Go into 'make menuconfig' (or whichever config is preferred) and set the following option:

Code:

```
Processor Type and Features ---->
High Memory Support ---->
```

(X) 4GB

This applies both to 2.4 and 2.6 kernels. Turning on high memory support theoretically slows down accesses slightly, but according to Joseph_sys and log, there is no practical difference.

The difference among VIRT, RES, and SHR in top output

VIRT stands for the virtual size of a process, which is the sum of memory it is actually using, memory it has mapped into itself (for instance the video card's RAM for the X server), files on disk that have been mapped into it (most notably shared libraries), and memory shared with other processes. VIRT represents how much memory the program is able to access at the present moment. RES stands for the resident size, which is an accurate representation of how much actual physical memory a process is consuming. (This also corresponds directly to the %MEM column.) This will virtually always be less than the VIRT size, since most programs depend on the C library.

SHR indicates how much of the VIRT size is actually sharable memory or libraries). In the case of libraries, it does not necessarily mean that the entire library is resident. For example, if a program only uses a few functions in a library, the whole library is mapped and will be counted in VIRT and SHR, but only the parts of the library file containing the functions being used will actually be loaded in and be counted under RES.

The difference between buffers and cache

Buffers are associated with a specific block device, and cover caching of filesystem metadata as well as tracking in-flight pages. The cache only contains parked file data. That is, the buffers remember what's in directories, what file permissions are, and keep track of what memory is being written from or read to for a particular block device. The cache only contains the contents of the files themselves.

Corrections and additions to this section welcome; I've done a bit of guesswork based on tracing how /proc/meminfo is produced to arrive at these conclusions.

Swappiness (2.6 kernels)

Since 2.6, there has been a way to tune how much Linux favors swapping out to disk compared to shrinking the caches when memory gets full.

ghoti adds: When an application needs memory and all the RAM is fully occupied, the kernel has two ways to free some memory at its disposal: it can either reduce the disk cache in the RAM by eliminating the oldest data or it may swap some less used portions (pages) of programs out to the swap partition on disk. It is not easy to predict which method would be more efficient. The kernel makes a choice by roughly guessing the effectiveness of the two methods at a given instant, based on the recent history of activity.

Before the 2.6 kernels, the user had no possible means to influence the calculations and there could happen situations where the kernel often made the wrong choice, leading to thrashing and slow performance. The addition of swappiness in 2.6 changes this. Thanks, ghoti!

Swappiness takes a value between 0 and 100 to change the balance between swapping applications and freeing cache. At 100, the kernel will always prefer to find inactive pages and swap them out; in other cases, whether a swapout occurs depends on how much application memory is in use and how poorly the cache is doing at finding and releasing inactive items.

The default swappiness is 60. A value of 0 gives something close to the old behavior where applications that wanted memory could shrink the cache to a tiny fraction of RAM. For laptops which would prefer to let their disk spin down, a value of 20 or less is recommended.

As a `sysctl`, the swappiness can be set at runtime with either of the following commands:

```
# sysctl -w vm.swappiness=30
# echo 30 >/proc/sys/vm/swappiness
```

The default when linux boots can also be set in `/etc/sysctl.conf`:

```
File: /etc/sysctl.conf
# Control how much the kernel should
#favor swapping out applications (0-100)
vm.swappiness = 30
```

Some patchsets allow the kernel to auto-tune the swappiness level as it sees fit; they may not keep a user-set value.