Linux 2.6 swapping behavior

Source : http://lwn.net/Articles/83588/

[Posted May 5, 2004 by corbet]

There has, recently, been a new round of complaints about how the 2.6 kernel swaps out memory. Some users have been very vocal in their belief that, if they have sufficient physical memory, their applications should never be swapped out. These people get annoyed when they sit down at their display in the morning and find that their office suite or web browser is unresponsive, and stays that way for some time. They get even more annoyed when they look and see how much memory the kernel is using for caching file contents rather than process memory. The obvious question to ask is: couldn't the kernel cut back a bit on the file caches and keep applications in memory?

The answer is that the kernel can be made to behave that way by tweaking a runtime parameter, but it is not necessarily a good idea. Before getting into that, however, it's worth noting that recent 2.6 kernels have a memory management problem which can cause serious problems after an application which reads through entire filesystems (updatedb, say, or a backup) has run. The problem is the slab cache's tendency to request allocations of multiple, contiguous pages; these allocations, when done at the behest of filesystem code, can bring the system to a halt. A patch has been merged which fixes this particular problem for 2.6.6.

The bigger issue remains, however: should the kernel swap out user applications in order to cache more file contents? There are plenty of arguments in favor of this behavior. Quite a few large applications set up big areas of memory which they rarely, if ever use. If application memory is occasionally forced to disk, the unused parts will remain there, and that much physical memory will be freed for more useful contents. Without swapping application memory to disk and seeing what gets faulted back in, it is almost impossible to figure out which pages are not really needed. A large file cache is also a performance enhancer. The speedups that come from having frequently-accessed data in memory are harder to see than the slowdowns caused by having to fault in a large application, but they can lead to better system throughput overall.

Still, there are users who insist that, for example, a system backup should never force OpenOffice out to disk. They don't care how quickly a system maintenance application runs at 3:00 in the morning, but they care a lot about how the system responds when they are at the keyboard. This wish was expressed repeatedly until Andrew Morton exclaimed:

I'm gonna stick my fingers in my ears and sing "la la la" until people tell me "I set swappiness to zero and it didn't do what I wanted it to do".

This helped quiet the debate as the parties involved looked more closely at this particular parameter. Or, perhaps, it was just fear of Andrew's singing. Either way, it has become clear that most people are unaware of what the "swappiness" parameter does; the fact that it has never been documented may have something to do with that.

So... swappiness, which is exported to **/proc/sys/vm/swappiness**, is a parameter which sets the kernel's balance between reclaiming pages from the page cache and swapping out process memory. The reclaim code works (in a very simplified way) by calculating a few numbers:

- The "**distress**" value is a measure of how much trouble the kernel is having freeing memory. The first time the kernel decides it needs to start reclaiming pages, distress will be zero; if more attempts are required, that value goes up, approaching a high value of 100.
- "**mapped_ratio**" is an approximate percentage of how much of the system's total memory is mapped (i.e. is part of a process's address space) within a given memory zone.
- "**vm_swappiness**" is the swappiness parameter, which is set to 60 by default.

With those numbers in hand, the kernel calculates its "**swap tendency**":

swap_tendency = mapped_ratio/2 + distress + vm_swappiness

If swap_tendency is below 100, the kernel will only reclaim page cache pages. Once it goes above that value, however, pages which are part of some process's address space will also be considered for reclaim. So, if life is easy, swappiness is set to 60, and distress is zero, the system will not swap process memory until it reaches 80% of the total. Users who would like to never see application memory swapped out can set swappiness to zero; that setting will cause the kernel to ignore process memory until the distress value gets quite high.

The swappiness parameter should do what a lot of users want, but it does not solve the whole problem. Swappiness is a global parameter; it affects every process on the system in the same way. What a number of people would like to see, however, is a way to single out individual applications for special treatment. Possible approaches include using the process's "nice" value to control memory behavior; a low-priority process would not be able to push out significant amounts of a high-priority process's memory. Alternatively, the VM subsystem and the scheduler could become more tightly integrated. The scheduler already makes an effort to detect "interactive" processes; those processes could be given the benefit of a larger working set in memory. That sort of thing is 2.7 work, however; in the mean time, people who are unhappy with the kernel's swap behavior may want to try playing with the knobs which have been provided.