

# Re: running Linux with no swap space (but lots of RAM)

---

*Source:* <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.system/2007-11/msg00224.html>

---

- *From:* Rainer Weikusat <[rweikusat@xxxxxxxxxxx](mailto:rweikusat@xxxxxxxxxxx)>
  - *Date:* Fri, 30 Nov 2007 16:13:46 +0100
- 

phil-news-nospam@xxxxxxxx writes:

On Thu, 29 Nov 2007 15:55:33 +0100 Rainer Weikusat <[rweikusat@xxxxxxxxxxx](mailto:rweikusat@xxxxxxxxxxx)> wrote:  
| phil-news-nospam@xxxxxxxx writes:

| [...]

|> |e> In that situation it is quite likely that you don't have "some  
|> |e> other device" to swap to. That's probably why you are booting  
|> |e> from flash in the first place.

|> | The problem in your new hypothetical is then that you have no device  
|> | capable of tolerating paging I/O, not that the system is paging to  
|> | disc.

|> Please explain what you mean by "tolerating paging I/O".

| In plain words, this means 'only devices whose actual behaviour and  
| limitations the OP understands even less well than he believes to  
| understand disks'.

| [...]

|> A frequent scenario I see happens when my need for memory by user space  
|> programs is below the capacity is available, and would not have even begun  
|> to swap anything. A program is run that will be doing a large amount of  
|> I/O output, such as copying 100+ GB of files between filesystems. The I/O  
|> buffering goes beyond just the pages that are free. The buffering logic  
|> tries to buffer far more of those 100+ GB than needed to keep the writing  
|> drive continuously busy or even to minimize head seeks.

| The purpose of the page cache is neither 'to keep the drive  
| continously busy' nor 'to minimize head seeks'. Both would be tasks the  
| elevator (or I/O-scheduler) is supposed to accomplish.

Some amount of caching is necessary to achieve such I/O scheduling.

Re: running Linux with no swap space (but lots of RAM)

For fairly obvious reasons, I/O-scheduling can only take place if there is actual something to schedule.

[...]

| Just for an informal test, I have just created an archive of all of my  
| filesystem to /dev/null (~16G). The amount of memory allocated to  
| 'buffers' and 'cached' (vmstat) peaked well below 95000K and 45000K,  
| respectively. No paging activities occurred during creation of the  
| archive.

Well of course no paging activities occurred during creation of the archive.  
This is a bogus test.

It is a test which should result in the amount of buffered disk contents growing 'without bounds', because all those files need to be read into memory.

[...]

| BTW, the by-and-far easiest path to personal happiness for you in this  
| respect is to just misconfigure your system to your hearts content  
| (its yours, after all) instead of talking about hypotheses you have  
| about situations which – for some strange reason – are not that  
| generally reproducible than the generality of your inferences would  
| require.

Someone who thinks writing to /dev/null would result in lots of data being queued for writing to the physical device is not someone I would care to consider technical advice from.

I didn't write that I assumed reading lots of disk files and writing their contents to /dev/null would result in lots of disk writes. From the point of view of the cache involved here, 'reading' and 'writing' make no difference: both cause memory pages to be filled with the contents of files which could later on be used to serve other read requests from memory or to collapse multiple writes to the same area of a file into a single disk write, because all except the last only wrote to the cached data.

| It would still be more sensible to add RAM until you don't experience  
| regular paging activities occurring for some unknown reason on your  
| system and leave the virtual memory configuration as-is to deal with  
| non-regular situations. Or consider reducing your working set.

## Re: running Linux with no swap space (but lots of RAM)

I already know that the amount of RAM to avoid the problem is too radical to consider. And I have yet to find a mainboard that supports 2 CPUs and 2 TB of RAM and fits in an ATX case using no more than 550 watts of power.

Despite your bogus test, the reality is that even with a working set well below the amount of actual RAM present, I/O writing (to real devices) will gradually force out pages.

Writing a 6.7G file on my 512M development machine did not do so. Not that a difference between reading and writing was to be expected in this respect.

A swapless system is a partial answer to this problem. My other idea is to have even more RAM, set up a big RAMFS, and at system initialization, copy /opt and /usr into there, and bind mount

that over the real /opt and /usr, in read-only mode. That would be another 8GB of RAM, bringing my system that would be well configured at 4GB in the usual way, up to 16GB (4GB for the original RAM need, 4GB more for swapless RAM space, and 8GB for /opt and /usr).

I really shouldn't have to structure the system to avoid this issue. There should be tuning knobs that allow me to reserve some portion of RAM (not a classification of particular RAM locations) to be used for process pages (e.g. program and library text pages), whether copied-on-write or not, that cannot under any circumstances ever be forced out due to lots of memory being used for write() buffering ...

The memory is used for content-caching under the assumption that this content will again be needed in future.

.