

Re: OT/drift: when is a RAMdisk an appropriate solution

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-08/msg05134.html

- *From:* Tom <Thomas-911@xxxxxxxxxxxxxxxx>
 - *Date:* Tue, 29 Aug 2006 20:18:44 GMT
-

On 29 Aug 2006 07:39:06 GMT, Chris Torek <nospam@xxxxxxxx> wrote:

Flash Gordon wrote:

[RAM-disk creation and manipulation is] no more difficult
and no
less system specific than it was back then.

Well, no less system-specific, to be sure. Sometimes "more (or less) difficult" depending on the system. (Aside: in vxWorks, just include the "ram disk" component in your project. The default name is "/ram0" but you can change it. If you want more than one, it is somewhat more difficult.)

The difference is that if you are doing it to speed things up
you
are quite possibly wasting your time and effort.

Indeed.

In article <[H5GIg.4036\\$y61.1915@fed1read05](mailto:H5GIg.4036$y61.1915@fed1read05)>
jmcgill <jmcgill@xxxxxxxxxxxxxxxxxxxxxxxx> wrote:

It still makes sense on diskless clients that boot over NFS and so on.
It's also indicated in some applications where security is sensitive, or
for devices that need to operate in environments where moving parts are
impractical.

Sometimes, a physical (as opposed to virtual, in-kernel-only) RAM
disk is appropriate. That is, you have a piece of hardware you
plug in, which looks and acts like a disk as far as the system is
concerned, but actually stores data in RAM.

Re: OT/drift: when is a RAMdisk an appropriate solution

While many systems (including vxWorks, as mentioned above) do include in-main-memory pseudo-disk devices, in a purely theoretical sense, there is **never** [%] any reason to use one: whatever you are doing that is going through some sort of "file system" layer, only to wind up going directly to regular old ordinary memory, is going through a "unnecessary" manipulation. File systems do a lot of work in order to deal with structured, and somewhat klunky, disk drive hardware, where memory exists in the form of "disk blocks" or "sectors" that can only be rewritten in entire units, cannot be moved or resized, and is generally very slow to access (about 5 or 6 decimal orders of magnitude slower than RAM). Real RAM does not share most of these characteristic drawbacks, and going through a (usually quite heavy) software layer that simply pretends to add some of them back is just silly.

[% One obvious exception is when the RAM-disk is being used for testing the file system software. Here, even in theory, the RAM drive is useful. :-)]

In practice, the reason for using a RAM disk is that the stubborn software (written by some stubborn and/or or long-gone programmer) insists on using the file system interface, and the system provides no way to short-circuit this. For instance, in C — to get at least marginally back on topic — one might have a program (or large library) that does all its I/O to a "stdio" stream. Since ISO C has no way to "open memory as a stream", one is often forced to supply an actual, on-disk file. To get decent performance, one may wish to use an "on-RAM-disk file" instead of "on-actual-disk".

Note that it can make sense to use a "RAM file system" rather than a "RAM disk" in this case. Going through a **real** file system, you may do a whole lot of extra work trying to avoid talking too much to a slow (real) disk — i.e., spend large amounts of CPU time to avoid waiting for the device. But if the "device" is fake, and actually is very fast (relative to real disks), this CPU time is being wasted. It may be better to go through a fake file system (one which uses RAM), instead of a real file system on a fake disk.

Of course, it would be even better still if C had some sort of memory-oriented stdio streams — or something like the 4.4BSD "funopen", which is even more general. (I am pretty sure that funopen() did get proposed for C99; but we might consider ourselves lucky to have gotten even snprintf(). :-))

Well written and detailed. Thank you Chris. I agree fully with you and wish to elaborate below a little to make sure I have not missed the boat.

I have revisited creating a very large array for holding raw data which is used in an iterative optimization algorithm. The purpose of

Re: OT/drift: when is a RAMdisk an appropriate solution

the large array is to eliminate the slow hard disk reads and increase program throughput. On an older machine (Win2000, Visual C++ 6.0, 256 M ram) when I compile with a huge array I get the following warning:

```
=====  
warning LNK4084: total image size 343744512 exceeds max (268435456);  
image may not run  
=====
```

The help doco on a newer machine (Win2000, Visual C++.NET, 1 G ram) indicates that the image can be as large as 1 G byte. (This machine is in the middle of a multi-day run and I do not have a compilation error/warning available at this time for this system.)

To get around the image size limits and to speed up I/O it seems a ram disk is one valid approach. A search of ram disk drivers shows that some can create disks 64+ Gig in size.

I am guessing on my older machine if I were to increase the ram that I would still be up against the same image size limitation. Thus the ram disk provides a great improvement for I/O for an iterative type process and allows you to work around image size limits?

I wish I knew how to load the data into a block of memory and have it treated as if were the cache for the hard drive. In other words, ready to use and no manipulation needed. No double handling: caching, paging, etc.

Chris's above reference to wishing for "funopen" combined with my image size problems leads me to believe that a ram disk is my only valid option at this time? Unless I were skilled enough to write my own funopen! That'll be the day!! :)

In the future the 64 bit machines may allow much larger image sizes? Then the data can loaded into arrays as long as memory resources are available. And only if the data is in nicely proportioned blocks (i.e. a data structure.) Arrays would not be the optimal solution for loose form, varying length or delimited type of data. In that case you really need to be able to assign a block of ram for I/O. I found the following 64 bit article interesting:

<http://msdn.microsoft.com/vstudio/java/compare/benchmark64/default.aspx>

Thus for my immediate needs and current inability to write my own "funopen" ... I must use a ram disk to speed up the I/O on my data hungry application?

Can anyone recommend their favorite vendor for a ram disk driver? There seem to be several available but I have yet to find a review on which is the best and most stable.

Re: OT/drift: when is a RAMdisk an appropriate solution

Thanks. -- Tom