

# Re: Binary data in multiple adjacent files

---

*Source:* <http://unix.derkeiler.com/Newsgroups/comp.os.vms/2006-10/msg02129.html>

---

- *From:* [briggs@xxxxxxxxxxxxxxxxxxxx](mailto:briggs@xxxxxxxxxxxxxxxxxxxx)
  - *Date:* 26 Oct 2006 10:22:57 -0500
- 

In article <45405A54.5AF48084@xxxxxxxxxxxx>, JF Mezei <jfmezei.spamnot@xxxxxxxxxxxx> writes:

I have a program that was setup to provide elevation data for all of australia new zealand from 2 large datasets that covered that territory. The application kpet the 2 files opened and did relative reads into those binary files to obtain the required data (and cached those record so that if the next request was for a nearby cell, it would save an IO).

Now however, updated data comes in much smaller files, each covering only 1° lat by 1° long. Each file has 1201 \* 1201 values, each value is 2 bytes. (basically elevation in metres for a 92 square metre area). Each file is roughly 2.8 megs.

So a raster line is  $1201 * 2 = 2402$  bytes long

Switching to this would mean that my application would need to open hundreds of files to cover australia for instance, so this introduces scalability issues. And since I need to review the code, I figured I should perhaps consider other methods.

For my current needs, I need to cover from 45 to 47° north, and from -73° to -76°, which gives me th following files:

N46W076 N46W075 N46W074  
N45W076 N45W075 N46W074

N46W074 covers 46°N 74°W at the lower left corner to 47°N 73°W at the upper right corner.

Looking at \$CRMPSC, it appears I need to do a RMS \$OPEN on each file and provide the channel associated with each file.

So this would eat into the FILLM quota.

## Re: Binary data in multiple adjacent files

Yes.

Now, if i just do the \$CRMPSC to map a whole file to some virtual address in my process space, would this consume 2,8 meg of working set right away ?

Mapping it doesn't eat any working set except, possibly, for the pages devoted to the page table entries that were updated when the section was mapped and which were, therefore, faulted into the working set. (Caveat: my internals knowledge is VAX-era. I don't know whether Alpha process PTE's count against process working set).

Or does VMS just allocate virtual memory that is marked invalid, and only when I try to access a few bytes at a location would VMS load a single page from the actual file that contain those 2 bytes I asked for ?

It's just mapped. It's not faulted into physical memory.

Yes, the two bytes come into your working set when you try to access them, and encounter a hard page fault and VMS reads them into physical memory.

You also get a whole "page fault cluster" faulted in. Check the SYSGEN "PFCDEFAULT" parameter or the "pfc" parameter to \$CRMPSC.

This is roughly analogous to RMS multi-block count. You read in multiple pages at one go because contiguous transfers are traditionally cheap and random seeks are traditionally expensive.

The nature of my application means that as I follow a route, I read data progressing in one direction, and the odds of having to go back are low (but not nil). So, blocks containing elevation data read early on are unlikely to be needed again once I have moved on to points outside those blocks.

Ok. So caching one block is a win. But retaining old blocks in the cache is near useless. And read-ahead or read-behind [sic] might be useful.

Does VMS provide a means where I can specify that I want at max 2 pages from a global section to mapped to my working set at any point in time ? (this way, when I request a shortword located in a different block, the system would automatically unmap the oldest block mapped to that file and use that memory to map the new block to the file).

## Re: Binary data in multiple adjacent files

No. Not that I know of.

The other thing I am thinking about is having say 5 files opened at any point in time, and wherever I access a file, I update some counter based on the progress in my processing. So when I need to open a new file, I would then close the file which has the lowest value in the counter (file that been idle for the longest).

Sounds reasonable. Though I'd probably go for 4 files as a max since the worst case is when your path hits a 4 way corner between intersecting regions and makes tiny circles around the intersection point. Two files would be a reasonable alternative, handling the case of a path that weaves across a boundary repeatedly but thrashing in the case of a path that winds around a border between three or more zones.

So, for every point, I would need to create a file name, and check in the list of currently opened file if that file has already been opened. But this would allow me to scale to any size.

Yes. Sounds like a plan.

Also, if I run this on 8.3 (Alpha): (written in C)

Say I do an fseek to the 824th byte in the file and then read 2 bytes. Then, I do an fseek to the 1020th byte and also read 2 bytes.

If you do an fseek to the 824th byte then the next seek should logically be to the 826th, 822nd, 3226th, 3228th, 3230th or somewhere in the next file back.

[Assumes that your application visits each 92 meter grid section between consecutive waypoints]

If you read in an 8192 byte page then you're likely to have two or three complete raster lines in memory already. That gives you the grid sections immediately north, south, northeast, northwest, southeast and southwest of your current location along with all the grid sections to your east and west to the edge of the file.

If you read in a 512 byte block then the only other relevant data you might have on hand are the grid sections to the east and west of your current position.

1. Do you want to read in additional pages so that you have ten or twenty

## Re: Binary data in multiple adjacent files

raster lines in memory?

2. If you read in "clusters" to make this happen, are you going to be happy with read-ahead, reading the target and a few subsequent blocks (to the south and east?) or are you going to want some read-behind as well (to the north and west?)

One obvious implementation technique would be to ditch VMS page fault handling and roll your own with RMS reads or \$QIO, reading in chunks of (let's say) 100 blocks at a time and always reading chunks aligned on 100 block boundaries.

100 blocks = 51200 bytes  $\approx$  24 raster lines. Now if you're traversing a file from north to south you can get away with 50 disk reads instead of 1201.

And if you're sticking to one grid line all the way east to west across the file, that's one disk read instead of four.

West to east traversal along a line of latitude is the only case where RMS, C or VMS page fault handling can hope to compete.

By aligning the reads on 100 block boundaries you tend to get effective read-ahead whether your path runs north to south or south to north. (You don't want to read 100 blocks starting at block 2000 then 100 blocks starting at block 1999 then 100 blocks starting at block 1998...)

Is the underlying IO system smart enough to know that both are accessing data from the same physical block and use the cache system ? Or is the C file IO so screwed up that it would bypass this facility ?

C file IO makes use of RMS buffer cache. My concern is that C/RMS random I/O is going to tend to fail to make use of the benefits available from larger block sizes. And the page fault cluster feature of VMS memory management is going to fail to optimize access patterns that proceed in reverse memory order.

.