

TIMING AND PROFILING - VIRTUAL MEMORY*

Charles Severance
Kevin Dowd

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 3.0[†]

In addition to the negative performance impact due to cache misses, the virtual memory system can also slow your program down if it is too large to fit in the memory of the system or is competing with other large jobs for scarce memory resources.

Under most UNIX implementations, the operating system automatically pages pieces of a program that are too large for the available memory out to the swap area. The program won't be tossed out completely; that only happens when memory gets extremely tight, or when your program has been inactive for a while. Rather, individual pages are placed in the swap area for later retrieval. First of all, you need to be aware that this is happening if you don't already know about it. Second, if it is happening, the memory access patterns are critical. When references are too widely scattered, your runtime will be completely dominated by disk I/O.

If you plan in advance, you can make a virtual memory system work for you when your program is too large for the physical memory on the machine. The techniques are exactly the same as those for tuning a software-managed out-of-core solution, or loop nests. The process of "blocking" memory references so that data consumed in neighborhoods uses a bigger portion of each virtual memory page before rotating it out to disk to make room for another.¹

1 Gauging the Size of Your Program and the Machine's Memory

How can you tell if you are running out-of-core? There are ways to check for paging on the machine, but perhaps the most straightforward check is to compare the size of your program against the amount of available memory. You do this with the `size` command:

```
% size myprogram
```

On a System V UNIX machine, the output looks something like this:

*Version 1.3: Aug 25, 2010 11:17 am -0500

[†]<http://creativecommons.org/licenses/by/3.0/>

¹We examine the techniques for blocking in Chapter 8.

$$53872 + 53460 + 10010772 = 10118104$$

On a Berkeley UNIX derivative you see something like this:

text	data	bss	hex	decimal
53872	53460	10010772	9a63d8	10118104

The first three fields describe the amount of memory required for three different portions of your program. The first, `text`, accounts for the machine instructions that make up your program. The second, `data`, includes initialized values in your program such as the contents of data statements, common blocks, externals, character strings, etc. The third component, `bss`, (block started by symbol), is usually the largest. It describes an uninitialized data area in your program. This area would be made of common blocks that are not set by a block data. The last field is a total for all three sections added together, in bytes.²

Next, you need to know how much memory you have in your system. Unfortunately, there isn't a standard UNIX command for this. On the RS/6000, `/etc/lscfg` tells you. On an SGI machine, `/etc/hinv` does it. Many System V UNIX implementations have an `/etc/memsize` command. On any Berkeley derivative, you can type:

```
% ps aux
```

This command gives you a listing of all the processes running on the machine. Find the process with the largest value in the `%MEM`. Divide the value in the `RSS` field by the percentage of memory used to get a rough figure for how much memory your machine has:

```
memory = RSS/(%MEM/100)
```

For instance, if the largest process shows 5% memory usage and a resident set size (RSS) of 840 KB, your machine has $840000/(5/100) = 16$ MB of memory.³ If the answer from the `size` command shows a total that is anywhere near the amount of memory you have, you stand a good chance of paging when you run — especially if you are doing other things on the machine at the same time.

2 Checking for Page Faults

Your system's performance monitoring tools tell you if programs are paging. Some paging is OK; page faults and "page-ins" occur naturally as programs run. Also, be careful if you are competing for system resources along with other users. The picture you get won't be the same as when you have the computer to yourself.

²Warning: The `size` command won't give you the full picture if your program allocates memory dynamically, or keeps data on the stack. This area is especially important for C programs and FORTRAN programs that create large arrays that are not in COMMON.

³You could also reboot the machine! It will tell you how much memory is available when it comes up.

To check for paging activity on a Berkeley UNIX derivative, use the `vmstat` command. Commonly people invoke it with a time increment so that it reports paging at regular intervals:

```
% vmstat 5
```

This command produces output every five seconds.

```
procs      memory
r b w    avm  fre  re at  pi  po  fr  de  sr s0 d1 d2 d3  in  sy  cs  us  sy  id
0 0 0    824 21568  0 0  0  0  0  0  0  0  0  0  20  37  13  0  1  98
0 0 0    840 21508  0 0  0  0  0  0  0  1  0  0  0 251 186 156  0 10 90
0 0 0    846 21460  0 0  0  0  0  0  0  2  0  0  0 248 149 152  1  9 89
0 0 0    918 21444  0 0  0  0  0  0  0  4  0  0  0 258 143 152  2 10 89
```

Lots of valuable information is produced. For our purposes, the important fields are `avm` or *active virtual memory*, the `fre` or *free real memory*, and the `pi` and `po` numbers showing paging activity. When the `fre` figure drops to near zero, and the `po` field shows a lot of activity, it's an indication that the memory system is overworked.

On a SysV machine, paging activity can be seen with the `sar` command:

```
% sar -r 5 5
```

This command shows you the amount of free memory and swap space presently available. If the free memory figure is low, you can assume that your program is paging:

```
Sat Apr 18 20:42:19
[r] freemem freeswap
    4032    82144
```

As we mentioned earlier, if you must run a job larger than the size of the memory on your machine, the same sort of advice that applied to conserving cache activity applies to paging activity.⁴ Try to minimize the stride in your code, and where you can't, blocking memory references helps a whole lot.

A note on memory performance monitoring tools: you should check with your workstation vendor to see what they have available beyond `vmstat` or `sar`. There may be much more sophisticated (and often graphical) tools that can help you understand how your program is using memory.

⁴By the way, are you getting the message "Out of memory?" If you are running `csh`, try typing `unlimit` to see if the message goes away. Otherwise, it may mean that you don't have enough swap space available to run the job.