Excerpted from the following, with permission:

Solaris® Troubleshooting Handbook

Troubleshooting and Performance Tuning Hints for Solaris® 10 and OpenSolaris®

Scott Cromar

Solaris® Troubleshooting Handbook
by Scott Cromar

ISBN-13: 978-1463512415

ISBN-10: 1463512414

Published by Cromar & Cromar
St Augustine, FL 32092

2011 Printing by CreateSpace, North Charleston, SC

Comments, questions, and corrections welcome at
http://solaristroubleshooting.blogspot.com

This book extends and supplements information first published at
http://www.princeton.edu/~unix/Solaris/troubleshoot

8

Memory and Paging

In the real world, memory shortfalls are much more devastating than having a CPU bottleneck. Two primary indicators of a RAM shortage are the scan rate and swap device activity. Table 8-1 shows some useful commands for monitoring both types of activity.

Table 8-1. Memory Monitoring Commands

Performance Metric	**Monitoring Commands**

Memory Saturation: Scan Rate	`sar -g`
	`vmstat`

Memory Saturation: Swap Space Usage and Paging Rates	`vmstat`
	`sar -g`
	`sar -p`
	`sar -r`
	`sar -w`

In both cases, the high activity rate can be due to something that does not have a consistently large impact on performance. The processes running on the system have to be examined to see how frequently they are run and what their impact is. It may be possible to re-work the program or run the process differently to reduce the amount of new data being read into memory.

(**Virtual memory** takes two shapes in a Unix system: physical memory and swap space. **Physical memory** usually comes in DIMM modules and is frequently called RAM. **Swap space** is a dedicated area of disk space that the operating system addresses almost as if it were physical memory. Since disk I/O is much slower than I/O to and from memory, we would prefer to use swap space as infrequently as possible. Memory **address space** refers to the range of addresses that can be assigned, or **mapped**, to virtual memory on the system. The bulk of an address space is not mapped at any given point in time.)

We have to weigh the costs and benefits of upgrading physical memory, especially to accommodate an infrequently scheduled process. If the cost is more important than the performance, we can use swap space to provide enough virtual memory space for the application to run. If adequate total virtual memory space is not provided, new processes will not be able to open. (The system may report "`Not enough space`" or "`WARNING: /tmp: File system full, swap space limit exceeded.`")

Swap space is usually only used when physical memory is too small to accommodate the system's memory requirements. At that time, space is freed in physical memory by **paging** (moving) it out to swap space. (See “Paging” below for a more complete discussion of the process.)

If inadequate physical memory is provided, the system will be so busy paging to swap that it will be unable to keep up with demand. (This state is known as "thrashing" and is characterized by heavy I/O on the swap device and horrendous performance. In this state, the scanner can use up to 80% of CPU.)

When this happens, we can use the **`vmstat -p`** command to examine whether the stress on the system is coming from executables, application data or file system traffic. This command displays the number of paging operations for each type of data.

Scan Rate

When available memory falls below certain thresholds, the system attempts to reclaim memory that is being used for other purposes. The **page scanner** is the program that runs through memory to see which pages can be made available by placing them on the free list. The **scan rate** is the number of times per second that the page scanner makes a pass through memory. (The "Paging" section later in this chapter discusses some details of the page scanner's operation.) The page scanning rate is the main tipoff that a system does not have enough physical memory. We can use **`sar -g`** or **`vmstat`** to look at the scan rate.

`vmstat 30` checks memory usage every 30 seconds. (Ignore the summary statistics on the first line.) If `page/sr` is much above zero for an extended time, your system may be running short of physical memory. (Shorter sampling periods may be used to get a feel for what is happening on a smaller time scale.)

A very low scan rate is a sure indicator that the system is not running short of physical memory. On the other hand, a high scan rate can be caused by transient issues, such as a process reading large amounts of uncached data. The processes on the system should be examined to see how much of a long-term impact they have on performance. Historical trends need to be examined with **`sar -g`** to make sure that the page scanner has not come on for a transient, non-recurring reason.

A nonzero scan rate is not necessarily an indication of a problem. Over time, memory is allocated for caching and other activities. Eventually, the amount of memory will reach the `lotsfree` memory level, and the `pageout` scanner will be invoked. For a more thorough discussion of the paging algorithm, see "Paging" below.

Swap Device Activity

The amount of disk activity on the swap device can be measured using **`iostat`**. **`iostat -xPnce`** provides information on disk activity on a partition-by-partition basis. **`sar -d`** provides similar information on a per-physical-device basis, and **`vmstat`** provides some usage information as well. Where Veritas Volume Manager is used, **`vxstat`** provides per-volume performance information.

If there are I/O's queued for the swap device, application paging is occurring. If there is significant, persistent, heavy I/O to the swap device, a RAM upgrade may be in order.

Process Memory Usage

The **`/usr/proc/bin/pmap`** command can help pin down which process is the memory hog. **`/usr/proc/bin/pmap -x`** ***`PID`*** prints out details of memory use by a process.

Summary statistics regarding process size can be found in the `RSS` column of **`ps -ly`** or **`top`**.

`dbx`, the debugging utility in the SunPro package, has extensive memory leak detection built in. The source code will need to be compiled with the -g flag by the appropriate SunPro compiler.

ipcs -mb shows memory statistics for shared memory. This may be useful when attempting to size memory to fit expected traffic.

Segmentation Violations

Segmentation violations occur when a process references a memory address not mapped by any segment. The resulting `SIGSEGV` signal originates as a major page fault hardware exception identified by the processor and is translated by `as_fault()` in the address space layer.

When a process overflows its stack, a segmentation violation fault results. The kernel recognizes the violation and can extend the stack size, up to a configurable limit. In a multithreaded environment, the kernel does not keep track of each user thread's stack, so it cannot perform this function. The thread itself is responsible for stack `SIGSEGV` (stack overflow signal) handling.

(The `SIGSEGV` signal is sent by the `threads` library when an attempt is made to write to a write-protected page just beyond the end of the stack. This page is allocated as part of the stack creation request.)

It is often the case that segmentation faults occur because of resource restrictions on the size of a process's stack. See "Resource Management" in Chapter 6 for information about how to increase these limits.

See "Process Virtual Memory" in Chapter 4 for a more detailed description of the structure of a process's address space.

Paging

Solaris uses both common types of paging in its virtual memory system. These types are **swapping** (swaps out all memory associated with a user process) and **demand paging** (swaps out the not recently used pages). Which method is used is determined by comparing the amount of available memory with several key parameters:

- **physmem**: `physmem` is the total page count of physical memory.
- **lotsfree**: The page scanner is woken up when available memory falls below `lotsfree`. The default value for this is `physmem`/64 (or 512 KB, whichever is greater); it can be tuned in the `/etc/system` file if necessary. The page scanner runs in demand paging mode by default. The initial scan rate is set by the kernel parameter `slowscan` (which is 100 by default).
- **minfree**: Between `lotsfree` and `minfree`, the scan rate increases linearly between `slowscan` and `fastscan`. (`fastscan` is determined experimentally by the system as the maximum scan rate that can be supported by the system hardware. `minfree` is set to `desfree`/2, and `desfree` is set to `lotsfree`/2 by default.) Each page scanner will run for `desscan` pages. This parameter is dynamically set based on the scan rate.
- **maxpgio**: `maxpgio` (default 40 or 60) limits the rate at which I/O is queued to the swap devices. It is set to 40 for x86 architectures and 60 for SPARC architectures. With modern hard drives, `maxpgio` can safely be set to 100 times the number of swap disks.
- **throttlefree**: When free memory falls below `throttlefree` (default `minfree`), the `page_create` routines force the calling process to wait until free pages are available.
- **pageout_reserve**: When free memory falls below this value (default `throttlefree`/2), only the page daemon and the scheduler are allowed memory allocations.

The **page scanner** operates by first freeing a usage flag on each page at a rate reported as "scan rate" in **`vmstat`** `and` **`sar -g`**. After `handspreadpages` additional pages have been read, the page scanner checks to see whether the usage flag has been reset. If not, the page is swapped out. (`handspreadpages` is set dynamically in current versions of Solaris. Its maximum value is `pageout_new_spread`.)

Solaris 8 introduced an improved algorithm for handling file system page caching (for file systems other than ZFS). This new architecture is known as the **cyclical page cache**. It is designed to remove most of the problems with virtual memory that were previously caused by the file system page cache.

In the new algorithm, the cache of unmapped/inactive file pages is located on a `cachelist` which functions as part of the `freelist`.

When a file page is mapped, it is mapped to the relevant page on the `cachelist` if it is already in memory. If the referenced page is not on the `cachelist`, it is mapped to a page on the `freelist` and the file page is read (or "paged") into memory. Either way, mapped pages are moved to the `segmap` file cache.

Once all other `freelist` pages are consumed, additional allocations are taken from the `cachelist` on a least recently accessed basis. With the new algorithm, file system cache only competes with itself for memory. It does not force applications to be swapped out of primary memory as sometimes happened with the earlier OS versions.

As a result of these changes, **`vmstat`** reports statistics that are more in line with our intuition. In particular, scan rates will be near zero unless there is a systemwide shortage of available memory. (In the past, scan rates would reflect file caching activity, which is not really relevant to memory shortfalls.)

Every active memory page in Solaris is associated with a **vnode** (which is a mapping to a file) and an **offset** (the location within that file). This references the **backing store** for the memory location, and may represent an area on the swap device, or it may represent a location in a file system. All pages that are associated with a valid `vnode` and offset are placed on the **global page hash list**.

`vmstat -p` reports paging activity details for applications (executables), data (anonymous) and file system activity.

The parameters listed above can be viewed and set dynamically via **`mdb`**, as in Example 8-1:

Example 8-1. Paging Parameters

```
# mdb -kw
Loading modules: [ unix krtld genunix specfs dtrace ufs sd ip sctp usba fcp fctl nca lofs zfs
random logindmux ptm cpc fcip sppp crypto nfs ]
> physmem/E
physmem:
physmem:        258887
> lotsfree/E
lotsfree:
lotsfree:       3984
> desfree/E
desfree:
desfree:        1992
> minfree/E
minfree:
minfree:        996
> throttlefree/E
throttlefree:
throttlefree:   996
> fastscan/E
fastscan:
fastscan:       127499
> slowscan/E
slowscan:
slowscan:       100
> handspreadpages/E
handspreadpages:
handspreadpages:127499
> pageout_new_spread/E
pageout_new_spread:
pageout_new_spread:             161760
> lotsfree/Z fa0
lotsfree:       0xf90                   =       0xfa0
```

```
> lotsfree/E
lotsfree:
lotsfree:       4000
```

Swap Space

The Solaris virtual memory system combines physical memory with available swap space via `swapfs`. If insufficient total virtual memory space is provided, new processes will be unable to open.

Swap space can be added, deleted or examined with the **`swap`** command. **`swap -l`** reports total and free space for each of the swap partitions or files that are available to the system. Note that this number does not reflect total available virtual memory space, since physical memory is not reflected in the output. **`swap -s`** reports the total available amount of virtual memory, as does **`sar -r`**.

If swap is mounted on `/tmp` via `tmpfs`, **`df -k /tmp`** will report on total available virtual memory space, both swap and physical. As large memory allocations are made, the amount of space available to `tmpfs` will decrease, meaning that the utilization percentages reported by **`df`** will be of limited use.

The DTrace Toolkit's **`swapinfo.d`** program prints out a summary of how virtual memory is currently being used. See Example 8-2:

Example 8-2. Virtual Memory Summary

```
# /opt/DTT/Bin/swapinfo.d
RAM  _______Total  2048 MB
RAM      Unusable    25 MB
RAM        Kernel   564 MB
RAM        Locked     2 MB
RAM          Used   189 MB
RAM          Free  1266 MB

Disk _______Total  4004 MB
Disk         Resv    69 MB
Disk        Avail  3935 MB

Swap _______Total  5207 MB
Swap         Resv    69 MB
Swap        Avail  5138 MB
Swap    (Minfree)   252 MB
```

Swapping

If the system is consistently below `desfree` of free memory (over a 30 second average), the **memory scheduler** will start to swap out processes. (ie, if both `avefree` and `avefree30` are less than `desfree`, the swapper begins to look at processes.)

Initially, the scheduler will look for processes that have been idle for `maxslp` seconds. (`maxslp` defaults to 20 seconds and can be tuned in `/etc/system`.) This swapping mode is known as **soft swapping**.

Swapping priorities are calculated for an LWP by the following formula:
`epri = swapin_time - rss/(maxpgio/2) - pri`
where `swapin_time` is the time since the thread was last swapped, `rss` is the amount of memory used by the LWPs process, and `pri` is the thread's priority.

If, in addition to being below `desfree` of free memory, there are two processes in the run queue and paging activity exceeds `maxpgio`, the system will commence **hard swapping**. In this state, the kernel unloads all modules and cache memory that is not currently active and starts swapping out processes sequentially until `desfree` of free memory is available.

Processes are not eligible for swapping if they are:

- In the SYS or RT scheduling class.
- Being executed or stopped by a signal.
- Exiting.
- Zombie.
- A system thread.
- Blocking a higher priority thread.

The DTrace Toolkit provides the **`anonpgpid.d`** script to attempt to identify the processes which are suffering the most when the system is hard swapping. While this may be interesting, if we are hard-swapping, we need to kill the culprit, not identify the victims. We are better off identifying which processes are consuming how much memory. **`prstat -s rss`** does a nice job of ranking processes by memory usage. (**RSS** stands for "**resident set size**, " which is the amount of physical memory allocated to a process.)

Example 8-3. Ranking Processes by Memory Usage

```
# prstat -s rss
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
   213 daemon     19M   18M sleep   59    0   0:00:12 0.0% nfsmapid/4
     7 root     9336K 8328K sleep   59    0   0:00:04 0.0% svc.startd/14
     9 root     9248K 8188K sleep   59    0   0:00:07 0.0% svc.configd/15
   517 root     9020K 5916K sleep   59    0   0:00:02 0.0% snmpd/1
   321 root     9364K 5676K sleep   59    0   0:00:02 0.0% fmd/14
...
Total: 39 processes, 159 lwps, load averages: 0.00, 0.00, 0.00
```

We may also find ourselves swapping if we are running `tmpfs` and someone places a large file in */tmp*. It takes some effort, but we have to educate our user community that */tmp* is *not* scratch space. It is literally part of the virtual memory space. It may help matters to set up a directory called */scratch* to allow people to unpack files or manipulate data.

System Memory Usage

`mdb` can be used to provide significant information about system memory usage. In particular, the **`::memstat`** dcmd, and the **`leak`** and **`leakbuf`** walkers may be useful.

- **`::memstat`** displays a memory usage summary. (See Example 8-4.)
- **`walk leak`** finds leaks with the same stack trace as a leaked `bufctl` or `vmem_seg`.
- **`walk leakbuf`** walks buffers for leaks with the same stack trace as a leaked `bufctl` or `vmem_seg`.

Example 8-4. System Memory Usage

```
> ::memstat
Page Summary                Pages                MB  %Tot
------------     ----------------  ----------------  ----
Kernel                      31563               246   12%
Anon                         1523                11    1%
Exec and libs                 416                 3    0%
Page cache                     70                 0    0%
Free (cachelist)            78487               613   30%
Free (freelist)            146828              1147   57%

Total                      258887              2022
Physical                   254998              1992
```

In addition, there are several functions of interest that can be monitored by DTrace:

Table 8-2. Memory Functions

Function Name	**Description**
page_exists()	Tests for a page with a given vnode and offset.
page_find()	Searches the hash list for a locked page that is known to have a given vnode and offset.
page_first()	Finds the first page on the global page hash list.
page_free()	Frees a page. If it has a vnode and offset, sent to the cachelist, otherwise sent to the freelist.
page_ismod()	Checks whether a page has been modified.
page_isref()	Checks whether a page has been referenced.
page_lock()	Lock a page structure.
page_lookup()	Find a page with the specified vnode and offset. If found on a free list, it will be moved from the freelist.
page_lookup_nowait()	Finds a page representing the specified vnode and offset that is not locked and is not on the freelist.
page_needfree()	Notifies the VM system that pages need to be freed.
page_next()	Next page on the global hash list.
page_release()	Unlock a page structure after unmapping it. Place it back on the cachelist if appropriate.
page_unlock()	Unlock a page structure.

Kernel Memory UsageSolaris kernel memory is used to provide space for kernel text, data and data structures. Most of the kernel's memory is nailed down and cannot be swapped.

For UltraSPARC and x64 systems, Solaris locks a translation mapping into the MMU's translation lookaside buffer (TLB) for the first 4MB of the kernel's text and data segments. By using large pages in this way, the number of kernel-related TLB entries is reduced, leaving more buffer resources for user code. This has resulted in tremendously improved performance for these environments.

When memory is allocated by the kernel, it is typically not released to the freelist unless a severe system memory shortfall occurs. If this happens, the kernel relinquishes any unused memory.

The kernel allocates memory to itself via the slab/kmem and vmem allocators. (A discussion of the internals of the allocators is beyond the scope of this book, but Chapter 11 of McDougall and Mauro discusses the allocators in detail.)

The kernel memory statistics can be tracked using **sar -k**, and probed using **mdb**'s **::kmastat** dcmd for an overall view of kernel memory allocation. The **kstat** utility allows us to examine a particular cache. Truncated versions of **::kmastat** and **kstat** output are demonstrated in Example 8-5:

Example 8-5. Kernel Memory Allocation

```
# mdb -k
Loading modules: [ unix krtld genunix specfs dtrace ufs sd ip sctp usba fcp fctl nca lofs zfs
random logindmux ptm cpc fcip sppp crypto nfs ]
> ::kmastat
cache                          buf    buf    buf    memory     alloc alloc
name                          size in use  total    in use   succeed  fail
```

```
------------------------- ------ ------ ------ --------- --------- -----
kmem_magazine_1                16    274   1016     16384      4569     0
...
bp_map_131072              131072      0      0         0         0     0
memseg_cache                  112      0      0         0         0     0
mod_hash_entries               24    187    678     16384    408634     0
...
thread_cache                  792    157    170    139264     75907     0
lwp_cache                     904    157    171    155648     11537     0
turnstile_cache                64    299    381     24576     86758     0
cred_cache                    148     50    106     16384     42752     0
rctl_cache                     40    586    812     32768    541859     0
rctl_val_cache                 64   1137   1651    106496   1148726     0
...
ufs_inode_cache               368  18526 102740  38256640    275296     0
...
process_cache                3040     38     56    172032     38758     0
...
zfs_znode_cache               192      0      0         0         0     0
------------------------- ------ ------ ------ --------- --------- -----
Total [static]                                    221184    150707     0
Total [hat_memload]                              7397376   8417187     0
Total [kmem_msb]                                 1236992    362278     0
Total [kmem_va]                                 42991616      8893     0
Total [kmem_default]                           152576000 112494417     0
Total [bp_map]                                    524288      3387     0
Total [kmem_tsb_default]                          319488     83391     0
Total [hat_memload1]                              245760    229486     0
Total [segkmem_ppa]                                16384       127     0
Total [umem_np]                                  1048576     11204     0
Total [segkp]                                   11010048     30423     0
Total [pcisch2_dvma]                              458752   8891868     0
Total [pcisch1_dvma]                               98304        11     0
Total [ip_minor_arena]                                64     13299     0
Total [spdsock]                                       64         1     0
Total [namefs_inodes]                                 64        21     0
------------------------- ------ ------ ------ --------- --------- -----

vmem                           memory     memory    memory     alloc alloc
name                           in use      total    import   succeed  fail
------------------------- --------- ---------- --------- --------- -----
heap                      1099614298112 4398046511104         0     20207     0
    vmem_metadata            6619136    6815744   6815744       752     0
        vmem_seg             5578752    5578752   5578752       681     0
        vmem_hash             722560     729088    729088        46     0
        vmem_vmem             295800     346096    311296       106     0
...
ibcm_local_sid                     0 4294967295         0         0     0
------------------------- --------- ---------- --------- --------- -----
> $Q

# kstat -n process_cache
module: unix                            instance: 0
name:   process_cache                   class:    kmem_cache
        align                           8
        alloc                           38785
        alloc_fail                      0
        buf_avail                       18
        buf_constructed                 12
        buf_inuse                       38
        buf_max                         64
        buf_size                        3040
        buf_total                       56
        chunk_size                      3040
        crtime                          28.796560304
        depot_alloc                     2955
        depot_contention                0
        depot_free                      2965
        empty_magazines                 0
        free                            38811
        full_magazines                  3
        hash_lookup_depth               1
        hash_rescale                    0
        hash_size                       64
        magazine_size                   3
        slab_alloc                      104
        slab_create                     9
        slab_destroy                    2
        slab_free                       54
        slab_size                       24576
        snaptime                        1233645.2648315
        vmem_source                     23
```

Certain aspects of the kernel memory allocation only become possible if the debug flags are enabled in **kmdb** at boot time, as in Example 8-6:

Example 8-6. Enabling Kernel Memory Allocator Debug Flag

```
ok boot kmdb -d
Loading kmdb...

Welcome to kmdb
[0]> kmem_flags/W 0x1f
kmem_flags:  0x0                =          0x1f
[0]> :c
```

If the system crashes while **kmdb** is loaded, it will drop to the **kmdb** prompt rather than the PROM monitor prompt. (This is intended to allow debugging to continue in the wake of a crash.) This is probably not the desired state for a production system, so it is recommended that **kmdb** be unloaded once debugging is complete.

0x1f sets all KMA flags. Individual flags can be set instead by using different values, but I have never run across a situation when it wasn't better to just have them all enabled.

Direct I/O

Large sequential I/O can cause performance problems due to excessive use of the memory page cache. One way to avoid this problem is to use direct I/O on file systems where large sequential I/Os are common.

Direct I/O is usually specified as a mount option in the *vfstab*. The specific file system option will vary based on file system type. For UFS, it is **forcedirectio**.

Resources

- Cockcroft, Adrian and Pettit, Richard. (April 1998) Sun Performance and Tuning: Java and the Internet, 2nd Ed. Prentice Hall.
- Cromar, Scott. (2007) *Solaris Troubleshooting and Performance Tuning at Princeton University*. Princeton, NJ. (http://www.princeton.edu/~unix/Solaris/troubleshoot/index.html)
- McDougall, Richard and Mauro, Jim. (July 2006) *Solaris Internals*. Upper Saddle River, NJ: Prentice Hall & Sun Microsystems Press.
- McDougall, Richard, Mauro, Jim and Gregg, Brendan. (October 2006) *Solaris Performance and Tools*. Upper Saddle River, NJ: Prentice Hall & Sun Microsystems Press.

- OpenSolaris Project. (October 2006) DTrace Toolkit. (http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/)
- Sun Microsystems. (May 2006) *Solaris Tunable Parameters Reference Manual*. Santa Clara, CA: Sun Microsystems, Inc. (http://docs.sun.com/app/docs/doc/817-0404)