```
Documentation for /proc/sys/vm/*        kernel version 2.6.29
        (c) 1998, 1999,  Rik van Riel <riel@nl.linux.org>
        (c) 2008         Peter W. Morreale <pmorreale@novell.com>

For general info and legal blurb, please look in README.

==============================================================

This file contains the documentation for the sysctl files in
/proc/sys/vm and is valid for Linux kernel version 2.6.29.

The files in this directory can be used to tune the operation
of the virtual memory (VM) subsystem of the Linux kernel and
the writeout of dirty data to disk.

Default values and initialization routines for most of these
files can be found in mm/swap.c.

Currently, these files are in /proc/sys/vm:

- block_dump
- compact_memory
- dirty_background_bytes
- dirty_background_ratio
- dirty_bytes
- dirty_expire_centisecs
- dirty_ratio
- dirty_writeback_centisecs
- drop_caches
- extfrag_threshold
- hugepages_treat_as_movable
- hugetlb_shm_group
- laptop_mode
- legacy_va_layout
- lowmem_reserve_ratio
- max_map_count
- memory_failure_early_kill
- memory_failure_recovery
- min_free_kbytes
- min_slab_ratio
- min_unmapped_ratio
- mmap_min_addr
- nr_hugepages
- nr_overcommit_hugepages
- nr_pdflush_threads
- nr_trim_pages         (only if CONFIG_MMU=n)
- numa_zonelist_order
- oom_dump_tasks
- oom_kill_allocating_task
- overcommit_memory
- overcommit_ratio
- page-cluster
- panic_on_oom
- percpu_pagelist_fraction
- stat_interval
- swappiness
```

- vfs_cache_pressure
- zone_reclaim_mode

==

block_dump

block_dump enables block I/O debugging when set to a nonzero value. More information on block I/O debugging is in Documentation/laptops/laptop-mode.txt.

==

compact_memory

Available only when CONFIG_COMPACTION is set. When 1 is written to the file, all zones are compacted such that free memory is available in contiguous blocks where possible. This can be important for example in the allocation of huge pages although processes will also directly compact memory as required.

==

dirty_background_bytes

Contains the amount of dirty memory at which the pdflush background writeback daemon will start writeback.

If dirty_background_bytes is written, dirty_background_ratio becomes a function of its value (dirty_background_bytes / the amount of dirtyable system memory).

==

dirty_background_ratio

Contains, as a percentage of total system memory, the number of pages at which the pdflush background writeback daemon will start writing out dirty data.

==

dirty_bytes

Contains the amount of dirty memory at which a process generating disk writes will itself start writeback.

If dirty_bytes is written, dirty_ratio becomes a function of its value (dirty_bytes / the amount of dirtyable system memory).

Note: the minimum value allowed for dirty_bytes is two pages (in bytes); any value lower than this limit will be ignored and the old configuration will be retained.

==

dirty_expire_centisecs

This tunable is used to define when dirty data is old enough to be eligible for writeout by the pdflush daemons. It is expressed in 100'ths of a second.

Data which has been dirty in-memory for longer than this interval will be written out next time a pdflush daemon wakes up.

==

dirty_ratio

Contains, as a percentage of total system memory, the number of pages at which a process which is generating disk writes will itself start writing out dirty data.

==

dirty_writeback_centisecs

The pdflush writeback daemons will periodically wake up and write `old' data out to disk. This tunable expresses the interval between those wakeups, in 100'ths of a second.

Setting this to zero disables periodic writeback altogether.

==

drop_caches

Writing to this will cause the kernel to drop clean caches, dentries and inodes from memory, causing that memory to become free.

```
To free pagecache:
        echo 1 > /proc/sys/vm/drop_caches
To free dentries and inodes:
        echo 2 > /proc/sys/vm/drop_caches
To free pagecache, dentries and inodes:
        echo 3 > /proc/sys/vm/drop_caches
```

As this is a non-destructive operation and dirty objects are not freeable, the user should run `sync' first.

==

extfrag_threshold

This parameter affects whether the kernel will compact memory or direct reclaim to satisfy a high-order allocation. /proc/extfrag_index shows what the fragmentation index for each order is in each zone in the system. Values tending towards 0 imply allocations would fail due to lack of memory, values towards 1000 imply failures are due to fragmentation and -1 implies that the allocation will succeed as long as watermarks are met.

The kernel will not compact memory in a zone if the fragmentation index is <= extfrag_threshold. The default value is 500.

==

hugepages_treat_as_movable

This parameter is only useful when kernelcore= is specified at boot time to create ZONE_MOVABLE for pages that may be reclaimed or migrated. Huge pages are not movable so are not normally allocated from ZONE_MOVABLE. A non-zero value written to hugepages_treat_as_movable allows huge pages to be allocated from ZONE_MOVABLE.

Once enabled, the ZONE_MOVABLE is treated as an area of memory the huge pages pool can easily grow or shrink within. Assuming that applications are not running that mlock() a lot of memory, it is likely the huge pages pool can grow to the size of ZONE_MOVABLE by repeatedly entering the desired value into nr_hugepages and triggering page reclaim.

==

hugetlb_shm_group

hugetlb_shm_group contains group id that is allowed to create SysV shared memory segment using hugetlb page.

==

laptop_mode

laptop_mode is a knob that controls "laptop mode". All the things that are controlled by this knob are discussed in Documentation/laptops/laptop-mode.txt.

==

legacy_va_layout

If non-zero, this sysctl disables the new 32-bit mmap mmap layout - the kernel will use the legacy (2.4) layout for all processes.

==

lowmem_reserve_ratio

For some specialised workloads on highmem machines it is dangerous for the kernel to allow process memory to be allocated from the "lowmem" zone. This is because that memory could then be pinned via the mlock() system call, or by unavailability of swapspace.

And on large highmem machines this lack of reclaimable lowmem memory can be fatal.

So the Linux page allocator has a mechanism which prevents allocations which _could_ use highmem from using too much lowmem. This means that a certain amount of lowmem is defended from the possibility of being captured into pinned user memory.

(The same argument applies to the old 16 megabyte ISA DMA region. This mechanism will also defend that region from allocations which could use highmem or lowmem).

The `lowmem_reserve_ratio' tunable determines how aggressive the kernel is in defending these lower zones.

If you have a machine which uses highmem or ISA DMA and your
applications are using mlock(), or if you are running with no swap then
you probably should change the lowmem_reserve_ratio setting.

The lowmem_reserve_ratio is an array. You can see them by reading this file.

```
-
% cat /proc/sys/vm/lowmem_reserve_ratio
256     256     32
-
```

Note: # of this elements is one fewer than number of zones. Because the highest
      zone's value is not necessary for following calculation.

But, these values are not used directly. The kernel calculates # of protection
pages for each zones from them. These are shown as array of protection pages
in /proc/zoneinfo like followings. (This is an example of x86-64 box).
Each zone has an array of protection pages like this.

```
-
Node 0, zone      DMA
  pages free     1355
        min      3
        low      3
        high     4
        :
        :
    numa_other   0
        protection: (0, 2004, 2004, 2004)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  pagesets
    cpu: 0 pcp: 0
        :
-
```

These protections are added to score to judge whether this zone should be used
for page allocation or should be reclaimed.

In this example, if normal pages (index=2) are required to this DMA zone and
watermark[WMARK_HIGH] is used for watermark, the kernel judges this zone should
not be used because pages_free(1355) is smaller than watermark + protection[2]
(4 + 2004 = 2008). If this protection value is 0, this zone would be used for
normal page requirement. If requirement is DMA zone(index=0), protection[0]
(=0) is used.

zone[i]'s protection[j] is calculated by following expression.

```
(i < j):
  zone[i]->protection[j]
  = (total sums of present_pages from zone[i+1] to zone[j] on the node)
    / lowmem_reserve_ratio[i];
(i = j):
   (should not be protected. = 0;
(i > j):
   (not necessary, but looks 0)
```

The default values of lowmem_reserve_ratio[i] are
    256 (if zone[i] means DMA or DMA32 zone)

    32 (others).
As above expression, they are reciprocal number of ratio.
256 means 1/256. # of protection pages becomes about "0.39%" of total present
pages of higher zones on the node.

If you would like to protect more pages, smaller values are effective.
The minimum value is 1 (1/1 -> 100%).

==

max_map_count:

This file contains the maximum number of memory map areas a process
may have. Memory map areas are used as a side-effect of calling
malloc, directly by mmap and mprotect, and also when loading shared
libraries.

While most applications need less than a thousand maps, certain
programs, particularly malloc debuggers, may consume lots of them,
e.g., up to one or two maps per allocation.

The default value is 65536.

===

memory_failure_early_kill:

Control how to kill processes when uncorrected memory error (typically
a 2bit error in a memory module) is detected in the background by hardware
that cannot be handled by the kernel. In some cases (like the page
still having a valid copy on disk) the kernel will handle the failure
transparently without affecting any applications. But if there is
no other uptodate copy of the data it will kill to prevent any data
corruptions from propagating.

1: Kill all processes that have the corrupted and not reloadable page mapped
as soon as the corruption is detected. Note this is not supported
for a few types of pages, like kernel internally allocated data or
the swap cache, but works for the majority of user pages.

0: Only unmap the corrupted page from all processes and only kill a process
who tries to access it.

The kill is done using a catchable SIGBUS with BUS_MCEERR_AO, so processes can
handle this if they want to.

This is only active on architectures/platforms with advanced machine
check handling and depends on the hardware capabilities.

Applications can override this setting individually with the PR_MCE_KILL prctl

==

memory_failure_recovery

Enable memory failure recovery (when supported by the platform)

1: Attempt recovery.

0: Always panic on a memory failure.

==

min_free_kbytes:

This is used to force the Linux VM to keep a minimum number
of kilobytes free. The VM uses this number to compute a
watermark[WMARK_MIN] value for each lowmem zone in the system.
Each lowmem zone gets a number of reserved free pages based
proportionally on its size.

Some minimal amount of memory is needed to satisfy PF_MEMALLOC
allocations; if you set this to lower than 1024KB, your system will
become subtly broken, and prone to deadlock under high loads.

Setting this too high will OOM your machine instantly.

===

min_slab_ratio:

This is available only on NUMA kernels.

A percentage of the total pages in each zone. On Zone reclaim
(fallback from the local zone occurs) slabs will be reclaimed if more
than this percentage of pages in a zone are reclaimable slab pages.
This insures that the slab growth stays under control even in NUMA
systems that rarely perform global reclaim.

The default is 5 percent.

Note that slab reclaim is triggered in a per zone / node fashion.
The process of reclaiming slab memory is currently not node specific
and may not be fast.

===

min_unmapped_ratio:

This is available only on NUMA kernels.

This is a percentage of the total pages in each zone. Zone reclaim will
only occur if more than this percentage of pages are in a state that
zone_reclaim_mode allows to be reclaimed.

If zone_reclaim_mode has the value 4 OR'd, then the percentage is compared
against all file-backed unmapped pages including swapcache pages and tmpfs
files. Otherwise, only unmapped pages backed by normal files but not tmpfs
files and similar are considered.

The default is 1 percent.

==

mmap_min_addr

This file indicates the amount of address space which a user process will be restricted from mmapping. Since kernel null dereference bugs could accidentally operate based on the information in the first couple of pages of memory userspace processes should not be allowed to write to them. By default this value is set to 0 and no protections will be enforced by the security module. Setting this value to something like 64k will allow the vast majority of applications to work correctly and provide defense in depth against future potential kernel bugs.

==

nr_hugepages

Change the minimum size of the hugepage pool.

See Documentation/vm/hugetlbpage.txt

==

nr_overcommit_hugepages

Change the maximum size of the hugepage pool. The maximum is nr_hugepages + nr_overcommit_hugepages.

See Documentation/vm/hugetlbpage.txt

==

nr_pdflush_threads

The current number of pdflush threads. This value is read-only.
The value changes according to the number of dirty pages in the system.

When necessary, additional pdflush threads are created, one per second, up to nr_pdflush_threads_max.

==

nr_trim_pages

This is available only on NOMMU kernels.

This value adjusts the excess page trimming behaviour of power-of-2 aligned NOMMU mmap allocations.

A value of 0 disables trimming of allocations entirely, while a value of 1 trims excess pages aggressively. Any value >= 1 acts as the watermark where trimming of allocations is initiated.

The default value is 1.

See Documentation/nommu-mmap.txt for more information.

==

numa_zonelist_order

This sysctl is only for NUMA.
'where the memory is allocated from' is controlled by zonelists.
(This documentation ignores ZONE_HIGHMEM/ZONE_DMA32 for simple explanation.
 you may be able to read ZONE_DMA as ZONE_DMA32...)

In non-NUMA case, a zonelist for GFP_KERNEL is ordered as following.
ZONE_NORMAL -> ZONE_DMA
This means that a memory allocation request for GFP_KERNEL will
get memory from ZONE_DMA only when ZONE_NORMAL is not available.

In NUMA case, you can think of following 2 types of order.
Assume 2 node NUMA and below is zonelist of Node(0)'s GFP_KERNEL

(A) Node(0) ZONE_NORMAL -> Node(0) ZONE_DMA -> Node(1) ZONE_NORMAL
(B) Node(0) ZONE_NORMAL -> Node(1) ZONE_NORMAL -> Node(0) ZONE_DMA.

Type(A) offers the best locality for processes on Node(0), but ZONE_DMA
will be used before ZONE_NORMAL exhaustion. This increases possibility of
out-of-memory(OOM) of ZONE_DMA because ZONE_DMA is tend to be small.

Type(B) cannot offer the best locality but is more robust against OOM of
the DMA zone.

Type(A) is called as "Node" order. Type (B) is "Zone" order.

"Node order" orders the zonelists by node, then by zone within each node.
Specify "[Nn]ode" for zone order

"Zone Order" orders the zonelists by zone type, then by node within each
zone. Specify "[Zz]one"for zode order.

Specify "[Dd]efault" to request automatic configuration. Autoconfiguration
will select "node" order in following case.
(1) if the DMA zone does not exist or
(2) if the DMA zone comprises greater than 50% of the available memory or
(3) if any node's DMA zone comprises greater than 60% of its local memory and
    the amount of local memory is big enough.

Otherwise, "zone" order will be selected. Default order is recommended unless
this is causing problems for your system/application.

==

oom_dump_tasks

Enables a system-wide task dump (excluding kernel threads) to be
produced when the kernel performs an OOM-killing and includes such
information as pid, uid, tgid, vm size, rss, cpu, oom_adj score, and
name. This is helpful to determine why the OOM killer was invoked
and to identify the rogue task that caused it.

If this is set to zero, this information is suppressed. On very large systems with thousands of tasks it may not be feasible to dump the memory state information for each one. Such systems should not be forced to incur a performance penalty in OOM conditions when the information may not be desired.

If this is set to non-zero, this information is shown whenever the OOM killer actually kills a memory-hogging task.

The default value is 0.

==

oom_kill_allocating_task

This enables or disables killing the OOM-triggering task in out-of-memory situations.

If this is set to zero, the OOM killer will scan through the entire tasklist and select a task based on heuristics to kill. This normally selects a rogue memory-hogging task that frees up a large amount of memory when killed.

If this is set to non-zero, the OOM killer simply kills the task that triggered the out-of-memory condition. This avoids the expensive tasklist scan.

If panic_on_oom is selected, it takes precedence over whatever value is used in oom_kill_allocating_task.

The default value is 0.

==

overcommit_memory:

This value contains a flag that enables memory overcommitment.

When this flag is 0, the kernel attempts to estimate the amount of free memory left when userspace requests more memory.

When this flag is 1, the kernel pretends there is always enough memory until it actually runs out.

When this flag is 2, the kernel uses a "never overcommit" policy that attempts to prevent any overcommit of memory.

This feature can be very useful because there are a lot of programs that malloc() huge amounts of memory "just-in-case" and don't use much of it.

The default value is 0.

See Documentation/vm/overcommit-accounting and security/commoncap.c::cap_vm_enough_memory() for more information.

==

overcommit_ratio:

When overcommit_memory is set to 2, the committed address space is not permitted to exceed swap plus this percentage of physical RAM. See above.

==

page-cluster

page-cluster controls the number of pages which are written to swap in a single attempt. The swap I/O size.

It is a logarithmic value - setting it to zero means "1 page", setting it to 1 means "2 pages", setting it to 2 means "4 pages", etc.

The default value is three (eight pages at a time). There may be some small benefits in tuning this to a different value if your workload is swap-intensive.

===

panic_on_oom

This enables or disables panic on out-of-memory feature.

If this is set to 0, the kernel will kill some rogue process, called oom_killer. Usually, oom_killer can kill rogue processes and system will survive.

If this is set to 1, the kernel panics when out-of-memory happens. However, if a process limits using nodes by mempolicy/cpusets, and those nodes become memory exhaustion status, one process may be killed by oom-killer. No panic occurs in this case. Because other nodes' memory may be free. This means system total status may be not fatal yet.

If this is set to 2, the kernel panics compulsorily even on the above-mentioned. Even oom happens under memory cgroup, the whole system panics.

The default value is 0.
1 and 2 are for failover of clustering. Please select either according to your policy of failover.
panic_on_oom=2+kdump gives you very strong tool to investigate why oom happens. You can get snapshot.

===

percpu_pagelist_fraction

This is the fraction of pages at most (high mark pcp->high) in each zone that are allocated for each per cpu page list. The min value for this is 8. It means that we don't allow more than 1/8th of pages in each zone to be allocated in any single per_cpu_pagelist. This entry only changes the value of hot per cpu pagelists. User can specify a number like 100 to allocate 1/100th of each zone to each per cpu page list.

The batch value of each per cpu pagelist is also updated as a result. It is set to pcp->high/4. The upper limit of batch is (PAGE_SHIFT * 8)

The initial value is zero. Kernel does not use this value at boot time to set the high water marks for each per cpu page list.

==

stat_interval

The time interval between which vm statistics are updated. The default is 1 second.

==

swappiness

This control is used to define how aggressive the kernel will swap memory pages. Higher values will increase agressiveness, lower values decrease the amount of swap.

The default value is 60.

==

vfs_cache_pressure

Controls the tendency of the kernel to reclaim the memory which is used for caching of directory and inode objects.

At the default value of vfs_cache_pressure=100 the kernel will attempt to reclaim dentries and inodes at a "fair" rate with respect to pagecache and swapcache reclaim. Decreasing vfs_cache_pressure causes the kernel to prefer to retain dentry and inode caches. When vfs_cache_pressure=0, the kernel will never reclaim dentries and inodes due to memory pressure and this can easily lead to out-of-memory conditions. Increasing vfs_cache_pressure beyond 100 causes the kernel to prefer to reclaim dentries and inodes.

==

zone_reclaim_mode:

Zone_reclaim_mode allows someone to set more or less aggressive approaches to reclaim memory when a zone runs out of memory. If it is set to zero then no zone reclaim occurs. Allocations will be satisfied from other zones / nodes in the system.

This is value ORed together of

```
1	= Zone reclaim on
2	= Zone reclaim writes dirty pages out
```

4   = Zone reclaim swaps pages

zone_reclaim_mode is set during bootup to 1 if it is determined that pages from remote zones will cause a measurable performance reduction. The page allocator will then reclaim easily reusable pages (those page cache pages that are currently not used) before allocating off node pages.

It may be beneficial to switch off zone reclaim if the system is used for a file server and all of memory should be used for caching files from disk. In that case the caching effect is more important than data locality.

Allowing zone reclaim to write out pages stops processes that are writing large amounts of data from dirtying pages on other nodes. Zone reclaim will write out dirty pages if a zone fills up and so effectively throttle the process. This may decrease the performance of a single process since it cannot use all of system memory to buffer the outgoing writes anymore but it preserve the memory on other nodes so that the performance of other processes running on other nodes will not be affected.

Allowing regular swap effectively restricts allocations to the local node unless explicitly overridden by memory policies or cpuset configurations.

============ End of Document =================================