

Linux内核参数介绍

讲师: 王攀

汇报时间: 2018-11-16

湖南麒麟信息工程技术有限公司





OOM机制-1

- OOM机制（ Out Of Memory killer ）
 - 当系统物理内存不够时，就会触发内核OOM机制，OOM机制根据一定的策略kill进程。从而避免因为内存不足导致kernel crash。
 - 老版本的kernel因为没有oom机制，经常出现因为内存不足导致系统“死机”。
- 与OOM相关的内核参数
 - vm.panic_on_oom
 - vm.oom_kill_allocating_task
 - vm.oom_dump_tasks
 - vm.would_have_oomkilled





OOM机制-2

- `vm.panic_on_oom = 0`
 - 0: oom被触发时, kernel不panic。若设置1, oom被触发, kernel便会panic
- `vm.oom_kill_allocating_task = 0`
 - 0: 不直接kill出发oom机制的进程, 而是在进程列表中挑选一个进程来kill, 一般选择耗费内存较大的进程来kill。
 - 设置成Non-zero:简单直接地kill出发oom机制的进程。/proc/[pid]/oom_score, 当前该pid进程的被kill的分数, 越高的分数意味着越可能被kill, 这个数值是根据oom_adj运算 (2^n, n就是oom_adj的值) 后的结果。oom_adj分数为15到-16, 其中15最大-16最小, -17为禁止使用OOM,分数越大越容易被kill, -17表示该进程会被保护, 不被kill。
- `vm.oom_dump_tasks = 1`
 - 1: 当oom触发时, 同时ulimit的core非0。那么被kill的进程会被coredump产生core文件。若0, 被kill的进程不会产生core文件。
- `vm.would_have_oomkilled = 0`
 - oom是否启用开关。默认是0, 表示启动。启动了oom, 在内存被耗尽时, 会kill进程, 系统处于正常运行状态。
 - 非零, 就是不启动。未启动oom, 在内存被耗尽时, 进程不会被kill, 但是系统整个就内存耗尽而卡死。



overcommit机制-1

- overcommit机制决定内存过量使用的策略
- 相关的内核参数
 - vm.overcommit_memory
 - vm.overcommit_ratio
 - vm.overcommit_kbytes





overcommit机制-2

- `vm.overcommit_memory = 0`
 - 该内核参数决定内存是否能被过量使用，默认是0。该值可以等于0，1，2，具体解释如下：
 - 0：启发式策略，就是一般的小量的过量分配是被允许的，如果大量的过量使用被禁止
 - 1：一直允许过量使用，不检查
 - 2：一直会被检查，不允许过量使用。检查的标准与`/proc/sys/vm/overcommit_ratio`相关，默认是50，在KYLIN 3.0平台是物理内存+50%的swap。在KYLIN 3.2上是（物理内存+swap）的50%。如果已经使用超过这个总和，后面的内存分配都是会失败。
- `vm.overcommit_ratio = 50`
 - 在`vm.overcommit_memory=2`时，这个参数才生效。单位是百分比。设置过量使用检查的百分比阈值。
- `vm.overcommit_kbytes = 0`
 - 与`vm.overcommit_ratio`作用一样。两个参数只用使用一个。一个被设置，另外一个会被disaabled。就会被设置成0



OOM与overcommit结合

- 在oom机制启用时，如果`vm.overcommit_memory = 0`，那么系统会100%使用完物理内存+swap时，才会kill进程。
- 如果oom机制启用时，如果`vm.overcommit_memory = 1`，那么系统同样会100%使用完物理内存+swap时，oom就会kill进程。
- 如果oom机制关闭了，并且`vm.overcommit_memory = 1`，那么系统物理内存耗尽，系统卡死。
- 如果即使oom机制关闭了，并且`vm.overcommit_memory = 2`，那么系统不会启用oom机制，但是内存如果使用到了物理内存+swap大小的50%（50%是默认配置，由`vm.overcommit_ratio = 50`来决定），那么进程就会分配内存失败，出现段错误。
- 如果oom机制启用了，并且`vm.overcommit_memory = 2`，系统虽然启用了oom机制，但是内存过度使用的检查先作用，所以先出现进程分配内存失败，段错误。oom只是在100%使用内存+swap时才起作用。

filesystem buffer机制-1

- Linux在处理进程的硬盘脏数据时，并非每一次的脏数据会立即从内存写入到硬盘中。而是使用内存建立了buffer机制。
- 与filesystem buffer机制相关的内核参数
 - vm.dirty_background_ratio
 - vm.dirty_background_bytes
 - vm.dirty_ratio
 - vm.dirty_bytes
 - vm.dirty_writeback_centisecs
 - vm.dirty_expire_centisecs

filesystem buffer机制-2

- `vm.dirty_background_ratio = 10`
 - 应用进程产生脏数据达到系统内存的10%时，系统就会启动pdflush进程把脏数据写回磁盘。
- `vm.dirty_background_bytes = 0`
 - 与`vm.dirty_background_ratio`作用相同，只是单位是bytes。但是两个参数只有一个生效。0就是不生效
- `vm.dirty_ratio = 20`
 - 应用进程产生脏数据达到系统内存的20%时，关闭buffer机制，让进程自行把脏数据写入硬盘。
- `vm.dirty_bytes = 0`
 - 与`vm.dirty_ratio`作用相同，只是单位是bytes。但是两个参数只有一个生效。0就是不生效
- `vm.dirty_writeback_centisecs = 500`
 - pdflush进程唤醒周期，单位是1/100秒。默认5s唤醒一次，将脏数据写入硬盘。
- `vm.dirty_expire_centisecs = 3000`
 - 如果脏数据在内存中驻留时间超过该值，pdflush进程在下一次将把这些数据写回磁盘。（单位是1/100秒）默认是30s



其它内存参数

- `vm.swappiness = 60`
 - 这个参数取值是0-100，决策的是交换分区如何使用。0表示，尽可能优先使用物理内存。100表示，尽可能将未使用的页缓存交换到硬盘的交换分区中。在优化设置过程中，一般都是设置成0或者1
- `vm.min_free_kbytes = 67584`
 - 系统会根据物理内存大小自动计算出一个内存大小。
 - 这个参数保留的物理内存是给内核使用的。如果系统剩余内存，低于这个内存值时，就会启动kswapd进程进行内存交换到交换分区上。这个数值一般不宜设置过大，设置过大导致系统频繁swap，也可能会触发oom机制。





socket快速回收

- `net.ipv4.tcp_tw_reuse = 1`
 - 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；
- `net.ipv4.tcp_tw_recycle = 1`
 - 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。
- `net.ipv4.tcp_fin_timeout = 30`
 - 表示如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2状态的时间。(单位：秒)





网络keepalive机制

- keepalive机制主要面向长连接应用场景，长连接在发包完毕后，会在一定的时间内保持连接，即我们通常所说的Keepalive（存活定时器）功能。麒麟操作系统为这个长连接提供了默认策略。就是长连接默认保持2个小时沉默，在这个2个小时中，若连续9次都没有收到探测报文的回复，那么就会断开本次的长连接。
- `net.ipv4.tcp_keepalive_time = 7200`
 - 默认7200秒。表示当keepalive起用的时候，长连接在应用程序没有任何数据通信的情况下，可以保持2个小时。并且一旦有数据通信，7200秒又得重新刷新计时。
- `net.ipv4.tcp_keepalive_probes = 9`
 - 在应用长连接开始启动7200s计时时，在连续9个探测空报文中，没有收到对方一个keepalive回复报文的情况下，系统会断开这个长连接。
- `net.ipv4.tcp_keepalive_intvl = 75`
 - 在应用长连接开始启动7200s计时时，每隔75s发送一次空报文进行存活判断。





网络连接控制

- `net.ipv4.tcp_syncookies = 1`
 - 表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；
- `net.ipv4.tcp_max_syn_backlog = 8192`
 - 表示SYN队列的长度，默认为1024，加大队列长度为8192，可以容纳更多等待连接的网络连接数。
- `net.ipv4.ip_local_port_range = 1024 65000` （端口号最大为65536）
 - 表示用于向外连接的端口范围。缺省情况下很小：32768到61000，改为1024到65000。
- `net.ipv4.ip_local_reserved_ports = 1,2-4,10-10, 8080,`
 - 预留本机的一些网络端口，避免客户端的网络链接占用





socket内存缓存

- `net.core.rmem_default = 262144`
–设置接收socket的缺省缓存大小(字节);
- `net.core.rmem_max = 4194304`
–设置接收socket的最大缓存大小(字节);
- `net.core.wmem_default = 262144`
–设置发送的socket缺省缓存大小(字节);
- `net.core.wmem_max = 4194304`
–设置发送的socket最大缓存大小(字节)
- `net.ipv4.tcp_wmem = 262144 262144 4194304`
–系统默认网络发送缓冲区大小值：第一个值是socket的发送缓存区分配的最少字节数,第二个值是默认值(该值会被`net.core.wmem_default`覆盖),缓存区在系统负载不重的情况下可以增长到这个值,第三个值是发送缓存区空间的最大字节数(该值会被`net.core.wmem_max`覆盖).
- `net.ipv4.tcp_rmem = 262144 262144 4194304`
–系统默认网络发送缓冲区大小值：三个值解释同上。





网络反向过滤

- `rp_filter`参数用于控制系统是否开启对数据包源地址的校验
 - `rp_filter`参数有三个值，0、1、2，具体含义：
 - 0：不开启源地址校验。
 - 1：开启严格的反向路径校验。对每个进来的数据包，校验其反向路径是否是最佳路径。如果反向路径不是最佳路径，则直接丢弃该数据包。
 - 2：开启松散的反向路径校验。对每个进来的数据包，校验其源地址是否可达，即反向路径是否能通（通过任意网口），如果反向路径不同，则直接丢弃该数据包。
- `accept_source_route` 参数控制是否接受含有源路由信息的ip包。
 - 参数值为布尔值，1表示接受，0表示不接受。在充当网关的linux主机上缺省值为1，在一般的linux主机上缺省值为0。从安全性角度出发，建议你关闭该功能。





进程数控制

- `kernel.threads-max = 93453`
 - 系统最多可以同时存在多少个线程数，不是针对单个进程的限制
- `kernel.pid_max = 32768`
 - 系统最多可以同时存在多少个进程数（说明：`ulimit -u`设置的每个用户最多可以同时存在多少个进程）
- `fs.file-max`
 - 设置系统所有进程所能打开文件数量的上限
- `fs.file-nr = 7712 0 590102`
 - file-nr包含三个数字，分别是系统当前分配了文件句柄的数量；释放的文件句柄数量；系统支持分配最大的文件句柄数量。
 - 系统支持的最大文件句柄数等于`fs.file-max`，随着`fs.file-max`的修改，`fs.file-nr`的第三个数字是会变为一样的。两个数字随着系统的运行是会动态变的。



共享内存设置

- `kernel.shmmax = 68719476736`
 - 设置单个共享内存段大小限制，如果进程创建共享内存大于这个限制时，系统会抛出这个错误，同时创建多个共享内存满足进程的需要。（单位：byte， $68719476736/1024/1024/1024=64\text{MB}$ ）
- `kernel.shmall`
 - 设置系统共享内存总大小，单位是页数。在x86体系结构下，内存页大小默认为4k。
 - 查看系统内存页大小命令：`getconf PAGE_SIZE`
 - 共享内存的大小与机器的物理内存大小没有直接关系。
- `kernel.shmmni`
 - 设置系统共享内存段的数量上限（单位是个数）



谢谢！