

# linux内核内存管理之oom

认真

# Table of Contents

- 1. 什么是OOM ..... 1
- 2. OOM造成的后果 ..... 1
- 3. 产生oom的原因..... 1
  - 3.1. UMA和NUMA介绍 ..... 1
  - 3.2. 参数panic\_on\_oom ..... 2
  - 3.3. oom原因判定 ..... 3
- 4. OOM有关的几个参数配置参考 ..... 4
  - 4.1. oom\_kill\_allocating\_task介绍 ..... 4
  - 4.2. oom\_dump\_tasks介绍 ..... 5
  - 4.3. 其他参数 ..... 6
- 5. 内存有关其它配置..... 8
  - 5.1. min\_free\_kbytes..... 8
  - 5.2. drop\_caches ..... 8
- 6. 总结..... 9
- 7. 状态..... 9

# 1. 什么是OOM

oom 是out of memory的缩写。对

linux而言，内核一般将处理器的虚拟地址空间划分为两个部分。底部比较大的部分用于用户进程，顶部专门用于内核。可用的物理内存被映射到内核的地址空间中，上层所有的进程不能直接操作物理内存，而是通过页表访问内存。

所以linux运行期间，需要不断的维护页表的申请，回收，再利用以保证程序可以有充足的可利用的内存。

那么，如果某个时刻，由于系统参数配置不当或者系统存在内存泄露等问题，造成了kernel分配page frame时候遇到内存耗尽，无法分配成功。

此时kernel遇到的情况，就是一种oom情形。

## 2. OOM造成的后果

- linux kernel 遇到了oom会怎么样？通常内核此时有两种选择：

- ① 系统崩溃并产生堆栈，就是常说的kernel panic,也就是立刻死机

- ② 内核会启动一个机制：OOM Killer，杀掉一些进程，试图继续运行

- OOM killer (Out-Of-Memory killer),该机制会监控系统中占用内存过大的进程，尤其是那些突然消耗非常大内存的进程，为了防止内存耗尽或者为了保证内核中重要进程有足够内存可以使用，会杀掉该进程，释放一部分内存出来。

## 3. 产生oom的原因

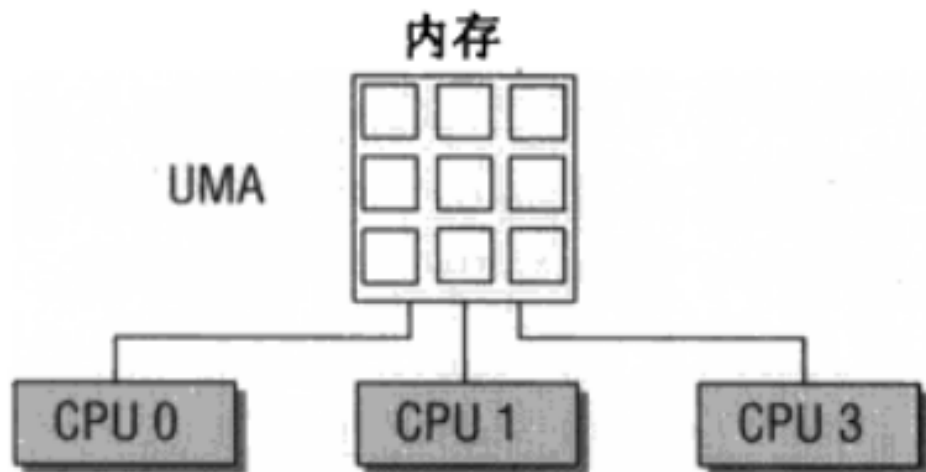


kernel产生oom，是否说明系统一定内存不足了呢？

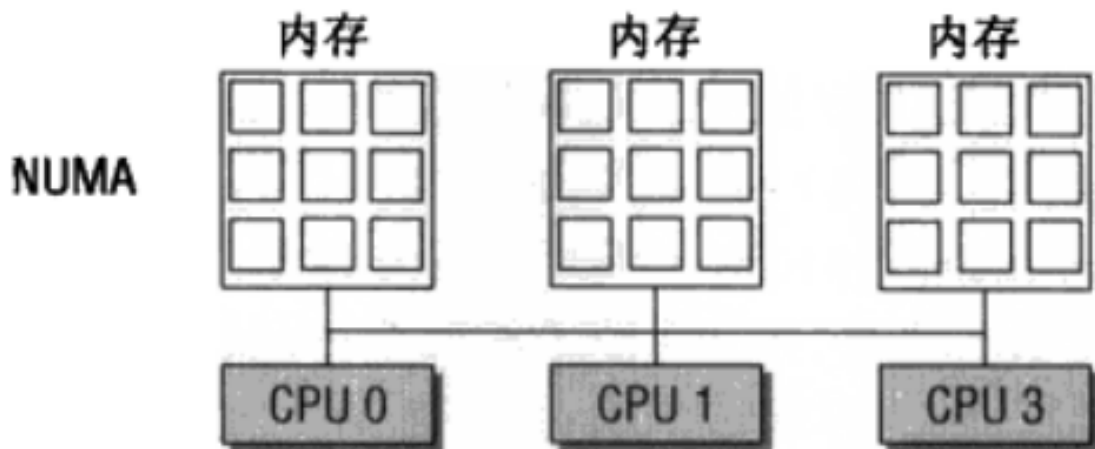
- 对于该问题的答案，要分情况来看待，对于 **UMA** 而言，一般就是内存不足了，但对于 **NUMA** 而言，并不一定。那么，究竟UMA和NUMA是什么？

### 3.1. UMA和NUMA介绍

- **UMA计算机**（一致内存访问，uniform memory access），这种类型内核将可用的内存以连续方式组织起来（也可能存在小洞情况以后再总结），在SMP系统中 每个CPU访问各自的内存区都是一样快。简单类似以下图示情况：



- **NUMA计算机**（非一致内存访问，non-uniform memory access）,这种类型总是多处理器计算机，系统的每个CPU都有本地内存，各个CPU之间通过总线连接起来 所以每个CPU都可以访问其它处理器的本地内存，但是访问速度上会慢一些。简单图示如下图：



### 3.2. 参数panic\_on\_oom

- panic\_on\_oom 位于系统 /proc/sys/vm/ 下，该值配置不同的数值，内核处理oom时就会有不同的策略
- 配置该参数，会调用内核函数接收配置的数值，决定启动不同的处理策略,内核接收函数如下图：

```

static void check_panic_on_oom(enum oom_constraint constraint, gfp_t gfp_mask,
                              int order, const nodemask_t *nodemask)
{
    if (likely(!sysctl_panic_on_oom))
        return;
    if (sysctl_panic_on_oom != 2) {
        /*
         * panic_on_oom == 1 only affects CONSTRAINT_NONE, the kernel
         * does not panic for cpuset, mempolicy, or memcg allocation
         * failures.
         */
        if (constraint != CONSTRAINT_NONE)
            return;
    }
    read_lock(&tasklist_lock);
    dump_header(NULL, gfp_mask, order, NULL, nodemask);
    read_unlock(&tasklist_lock);
    panic("Out of memory: %s panic_on_oom is enabled\n",
          sysctl_panic_on_oom == 2 ? "compulsory" : "system-wide");
}

```

--:--- oom\_kill.c [?] 67% (542,0) <N> Git-d43c476 (C/l Ifdef Hiding GG HL hl-s hl-p  
Undo branch point!

- 不同的参数值，表示内核遇到oom时，应当如何处置：
  - panic\_on\_oom = 0 直接返回，相当于开启了oom\_killer机制
  - panic\_on\_oom = 1 并且没有配置无约束标志 **CONSTRAINT\_NONE** ,可以尝试oom\_killer。但是UMA系统，该处总是 **CONSTRAINT\_NONE**
  - panic\_on\_oom = 2 直接panic

### 3.3. oom原因判定

- 内核中有四个标志性变量，分别表示一定的约束条件，如下图所示：

```

enum oom_constraint {
    CONSTRAINT_NONE,
    CONSTRAINT_CPUSET,
    CONSTRAINT_MEMORY_POLICY,
    CONSTRAINT_MEMCG,
};

```

- 每个标志位的含义解释如下，需要说明的是，下列配置选项仅针对 **NUMA** :
  - **CONSTRAINT\_CPUSET** cpuset是linux kernel的一种机制，该机制可以把一组cpu和memory node分

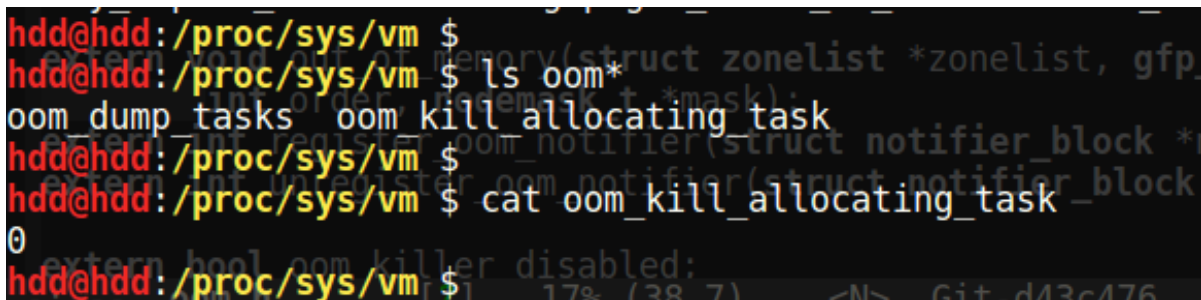
配给特定的

一组进程。如果此时出现了OOM，仅仅说明该进程能分配memory的那个node出现问题，此时系统还有很多内存。

- `CONSTRAINT_MEMORY_POLICY` memory policy是NUMA系统中，如何控制分配各个memory node资源的一个策略模块。产生OOM，也可能是该memory policy出现问题导致。
  - `CONSTRAINT_MEMCG` memory control group,简单说就是控制系统memory分配的控制器。该控制器可以将一组进程 内存使用限制在一个范围呢，如果超出该范围，就会出现 OOM
- 综上，对于UMA系统，出现OOM,一般就是内存不足了。但是对于NUMA系统，出现OOM,或许系统还有充足的内存，具体原因就要进一步分析了。

## 4. OOM有关的几个参数配置参考

- 在系统路径 `/proc/sys/vm` 下，可以看到关于oom的有两个很关键的参数，如下图所示：



```
hdd@hdd:/proc/sys/vm $  
hdd@hdd:/proc/sys/vm $ ls oom*  
oom_dump_tasks oom_kill_allocating_task  
hdd@hdd:/proc/sys/vm $  
hdd@hdd:/proc/sys/vm $ cat oom_kill_allocating_task  
0  
hdd@hdd:/proc/sys/vm $
```

那么，这两个参数到底有什么用途呢？

### 4.1. oom\_kill\_allocating\_task介绍

- 该参数 `oom_kill_allocating_task` 是决定系统产生oom后，oom\_killer机制启动，内核到底可以杀掉哪个进程。配置不同的数值（0/1），内核可以有两个选择
  - **1** --哪个进程触发了oom，就干掉它
  - **0** --根据一定算法计算出，此刻哪个进程“得分”最高，干掉它
  - 代码依据如下图所示：

```
4.3. oom原因判定
/* Check if there were limitations on the allocation (only relevant for
   OOM有关的几个参数配置参考
4.1. oom_kill_allocating_task介绍
4.2. oom_dump_tasks介绍
constraint = constrained_alloc(zonelist, gfp_mask, nodemask,
                                &totalpages);
mpol_mask = (constraint == CONSTRAINT_MEMORY_POLICY) ? nodemask : NULL;
check_panic_on_oom(constraint, gfp_mask, order, mpol_mask); //用途呢?

read_lock(&tasklist_lock);
if (sysctl_oom_kill_allocating_task && 4.1. oom_kill_allocating_task介绍
    !oom_unkillable_task(current, NULL, nodemask) &&
    current->mm) {
    /*
     * oom_kill_process() needs tasklist lock held. If it returns
     * non-zero, current could not be killed, so we must fallback to
     * the tasklist scan.
     */
    if (!oom_kill_process(current, gfp_mask, order, 0, totalpages,
                          NULL, nodemask,
                          "Out of memory (oom_kill_allocating_task)"))
        goto out;
}

完成BUG
```

```
the limitations on the allocation (only relevant for
require different handling.
hed_alloc(zonelist, gfp_mask, nodemask,
totalpages);
nt == CONSTRAINT_MEMORY_POLICY) ? nodemask : NULL;
straint, gfp_mask, order, mpol_mask); //用途呢?

ock);
llocating_task && 4.1. oom_kill_allocating_task介绍
sk(current, NULL, nodemask) &&
    • 该参数 oom_kill_allocating_task 是决定系统产生
      (0/1) 内核可以有两个选择
ss() needs tasklist lock held. If it returns
ent could not be killed, so we must fallback to
can.
    • 0 根据一定算法计算出, 此刻哪个进程“得分”最高
    • 1 哪个进程触发了OOM, 就干掉它
ess(current, gfp_mask, order, 0, totalpages,
emask,
emory (oom_kill_allocating_task)))
4.2. oom_dump_tasks介绍

完成BUG
```

## 4.2. oom\_dump\_tasks介绍

- 该参数 oom\_dump\_task 可以配置 0 或者 1, 主要是系统产生 oom 时候, 是否要收集输出一些进程信息。
  - 0 -- 不会打印出该信息
  - 1 -- 要去收集进程关于内存方面的信息并且打出, 方便找出 oom 具体原因

## 4.3. 其他参数

- 常见的其他进程相关的oom参数有以下几个
  - oom\_adj
  - oom\_score
  - oom\_score\_adj
- 这些参数位于 /proc/PID/ 下，其中PID指的是进程ID,如下图

```
hdd@hdd:/proc $ cd 1658
hdd@hdd:/proc/1658 $ ls
attr          cgroup        {..comm       cwd
autogroup     clear_refs    coredump_filter environ
auxv          cmdline       cpuset
hdd@hdd:/proc/1658 $ ls oom*
oom_adj       oom_score     oom_score_adj
hdd@hdd:/proc/1658 $ cat oom_adj
0
hdd@hdd:/proc/1658 $ cat oom_score
1
hdd@hdd:/proc/1658 $ cat oom_score_adj
0
hdd@hdd:/proc/1658 $
```



```

$ unsigned long oom_badness(str
$ cd 1658
1658 $ ls
1658 $ ls oom*
1658 $ cat oom_adj
1658 $ cat oom_score
1658 $ cat oom_score_adj
1658 $

```

- 这些参数又有什么用呢？当系统出现oom时，又配置内核选择oom\_killer机制时，内核会根据算法给每个进程一个 oom\_score，分数越高，被内核杀掉的几率越高。然而这个oom\_score是根据oom\_obj计算出来的，oom\_obj是可以配置的，配置范围为15 ~ -16 之间，配置-17则说明，禁止使用oom\_killer，参见以下内核代码：

```

#define OOM_SCORE_ADJ_MIN (-1000)
#define OOM_SCORE_ADJ_MAX 1000
/* /proc/<pid>/oom_adj is deprecated, see
 * Documentation/feature-removal-schedule.txt.
 * /proc/<pid>/oom_adj set to -17 protects from the oom-killer
 */
#include <linux/sched.h>
#include <linux/types.h>
#include <linux/nodemask.h>
#define OOM_DISABLE (-17)
/* inclusive */
#define OOM_ADJUST_MIN (-16)
#define OOM_ADJUST_MAX 15
struct notifier_block;
struct mem_cgroup;
struct task_struct;
/* /proc/<pid>/oom_score_adj set to OOM_SCORE_ADJ_MIN disables oom killing for
 * pid.
 */
/* Types of limitations to the nodes from which allocations may occur */
#define OOM_SCORE_ADJ_MIN (-1000)
#define OOM_SCORE_ADJ_MAX 1000
enum oom_constraint {
    OOM_CONSTRAINT_NONE = 0,
    OOM_CONSTRAINT_NODE = 1,
    OOM_CONSTRAINT_MIGRATION = 2,
    OOM_CONSTRAINT_LOCAL = 3,
    OOM_CONSTRAINT_REMOTE = 4,
    OOM_CONSTRAINT_ALL = 5,
    OOM_CONSTRAINT_MAX = 5
};

```

- 所以，如果要配置一个进程不被oom\_kill，则可以参照下面一个配置例子,以ssh为例:

```
pgrep -f "sshd" | while read PID; do echo -17 > /proc/$PID/oom_adj;done
```

## 5. 内存有关其它配置

### 5.1. min\_free\_kbytes

- 该参数 min\_free\_kbytes,位于系统目录 /proc/sys/vm 下
  - 该值保证系统间可用的最小KB数，这个值被内核用来计算每个低内存区的水印值，然后为其大小按照比例分配保留可用页。



min\_free\_kbytes 这个值配置时候一定要非常慎重，因为该值配置过高或者过低都会很大风险。如果该值配置太低，导致系统挂起并触发oom的killer杀死多个进程。之前的TA98设备该值过低，导致powerap不断被杀掉导致AP离线。如果该值配置过高(占系统总内存的5%-10%),会造成系统很快内存不足。

- 以下图示说明了系统总内存和 min\_free\_kbytes最低值的对应关系参考:

主内存大小	保留内存大小
16 MiB	512 KiB
32 MiB	724 KiB
64 MiB	1024 KiB
128 MiB	1448 KiB
256 MiB	2048 KiB
512 MiB	2896 KiB
1024 MiB	4096 KiB
2048 MiB	5792 KiB
4096 MiB	8192 KiB
8192 MiB	11584 KiB
16384 MiB	16384 KiB

### 5.2. drop\_caches

- 该参数 drop\_caches ,同样位于目录 /proc/sys/vm 下
  - 该值的配置，参见下图，下图是linux内核中文档关于该参数配置说明

```

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.

```

- 通过查看内核文档，可以看出
  - drop\_caches=1 系统释放所有页缓冲内存
  - drop\_caches=2 系统释放所有未使用的slab缓冲内存
  - drop\_caches=3 系统释放所有页缓冲内存和slab缓冲内存



利用sysctl命令配置该参数为 0，禁止内核清理缓存。对于我们的AP设备，由于重点在于网络数据的转发，所以该参数最好配置为 0，缓冲数据有利于报文的转发，提升转发效率。

## 6. 总结

以上是自己的一点总结，参考来自书籍，网络，内核文档，内核代码。  
当然，毕竟是自己现阶段的理解，也可能对有些概念的阐述存在不当之处。  
希望以上的总结对大家有一点的帮助。

## 7. 状态

- ☒ 初稿完成
- ☒ 继续完善，修改，总结