



下载APP



## 08 案例篇 | Shmem: 进程没有消耗内存，内存哪去了？

2020-09-05 邵亚方

Linux内核技术实战课

[进入课程 >](#)**讲述：邵亚方**

时长 12:05 大小 11.08M



你好，我是邵亚方。

在前一节课，我们讲述了进程堆内存的泄漏以及因为内存泄漏而导致的 OOM 的危害。这节课我们继续讲其他类型的内存泄漏，这样你在发现系统内存越来越少时，就能够想到会是什么在消耗内存。

有的内存泄漏会体现在进程内存里面，这种相对好观察些；而有的内存泄漏就很难观察了，因为它们无法通过观察进程消耗的内存来进行判断，从而容易被忽视，比如 Shmem 内存泄漏就属于这种容易被忽视的，这节课我们重点来讲讲它。



### 进程没有消耗内存，内存哪去了？

我生产环境上就遇到过一个真实的案例。我们的运维人员发现某几台机器 used（已使用的）内存越来越多，但是通过 top 以及其他一些命令，却检查不出来到底是谁在占用内存。随着可用内存变得越来越少，业务进程也被 OOM killer 给杀掉，这给业务带来了比较严重的影响。于是他们向我寻求帮助，看看产生问题的原因是什么。

我在之前的课程中也提到过，在遇到系统内存不足时，我们首先要做的是查看 /proc/meminfo 中哪些内存类型消耗较多，然后再去做针对性分析。但是如果你不清楚 /proc/meminfo 里面每一项的含义，即使知道了哪几项内存出现了异常，也不清楚该如何继续去分析。所以你最好是记住 /proc/meminfo 里每一项的含义。


回到我们这个案例，通过查看这几台服务器的 /proc/meminfo，发现 Shmem 的大小有些异常：

 复制代码

```
1 $ cat /proc/meminfo
2 ...
3 Shmem 16777216 kB
4 ...
```

那么 Shmem 这一项究竟是什么含义呢？该如何去进一步分析到底是谁在使用 Shmem 呢？

我们在前面的基础篇里提到，Shmem 是指匿名共享内存，即进程以 mmap（MAP\_ANON|MAP\_SHARED）这种方式来申请的内存。你可能会有疑问，进程以这种方式来申请的内存不应该是属于进程的 RES（resident）吗？比如下面这个简单的示例：

 复制代码

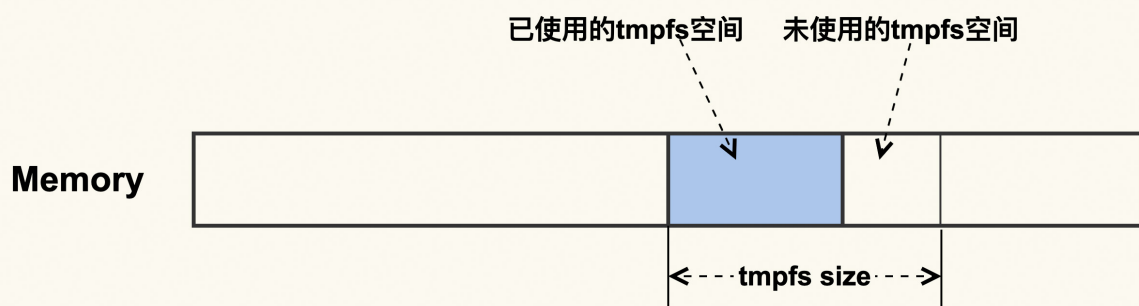
```
1 #include <sys/mman.h>
2 #include <string.h>
3 #include <unistd.h>
4 #define SIZE (1024*1024*1024)
5
6 int main()
7 {
8     char *p;
9
10    p = mmap(NULL, SIZE, PROT_READ|PROT_WRITE, MAP_ANON|MAP_SHARED, -1, 0)
```

```
11     if (!p)
12         return -1;
13
14     memset(p, 1, SIZE);
15
16     while (1) {
17         sleep(1);
18     }
19
20     return 0;
21 }
```

运行该程序后, 通过 top 可以看到确实会体现在进程的 RES 里面, 而且还同时体现在了进程的 SHR 里面, 也就是说, 如果进程是以 mmap 这种方式来申请内存的话, 我们是可以观察到进程的内存消耗来观察到的。


但是在我们生产环境上遇到的问题, 各个进程的 RES 都不大, 看起来和 /proc/meminfo 中的 Shmem 完全对应不起来, 这又是为什么呢?

先说答案: 这跟一种特殊的 Shmem 有关。我们知道, 磁盘的速度是远远低于内存的, 有些应用程序为了提升性能, 会避免将一些无需持续化存储的数据写入到磁盘, 而是把这部分临时数据写入到内存中, 然后定期或者在不需要这部分数据时, 清理掉这部分内容来释放出内存。在这种需求下, 就产生了一种特殊的 Shmem: tmpfs。tmpfs 如下图所示:



它是一种内存文件系统, 只存在于内存中, 它无需应用程序去申请和释放内存, 而是操作系统自动来规划好一部分空间, 应用程序只需要往这里面写入数据就可以了, 这样会很方便。我们可以使用 moun 命令或者 df 命令来看系统中 tmpfs 的挂载点:

```
1 $ df -h
2 Filesystem      Size  Used Avail Use% Mounted on
3 ...
4 tmpfs           16G   15G   1G   94% /run
5 ...
```

 复制代码


就像进程往磁盘写文件一样, 进程写完文件之后就把文件给关闭掉了, 这些文件和进程也就不再有关联, 所以这些磁盘文件的大小不会体现在进程中。同样地, tmpfs 中的文件也一样, 它也不会体现在进程的内存占用上。讲到这里, 你大概已经猜到了, 我们 Shmem 占用内存多, 是不是因为 Shmem 中的 tmpfs 较大导致的呢?

tmpfs 是属于文件系统的一种。对于文件系统, 我们都可以通过 df 来查看它的使用情况。所以呢, 我们也可以通过 df 来看是不是 tmpfs 占用的内存较多, 结果发现确实是它消耗了很多内存。这个问题就变得很清晰了, 我们只要去分析 tmpfs 中存储的是什么文件就可以了。

我们在生产环境上还遇到过这样一个问题: systemd 不停地往 tmpfs 中写入日志但是没有去及时清理, 而 tmpfs 配置的初始值又太大, 这就导致 systemd 产生的日志量越来越多, 最终可用内存越来越少。

针对这个问题, 解决方案就是限制 systemd 所使用的 tmpfs 的大小, 在日志量达到 tmpfs 大小限制时, 自动地清理掉临时日志, 或者定期清理掉这部分日志, 这都可以通过 systemd 的配置文件来做到。tmpfs 的大小可以通过如下命令 (比如调整为 2G) 调整:

```
1 $ mount -o remount,size=2G /run
```

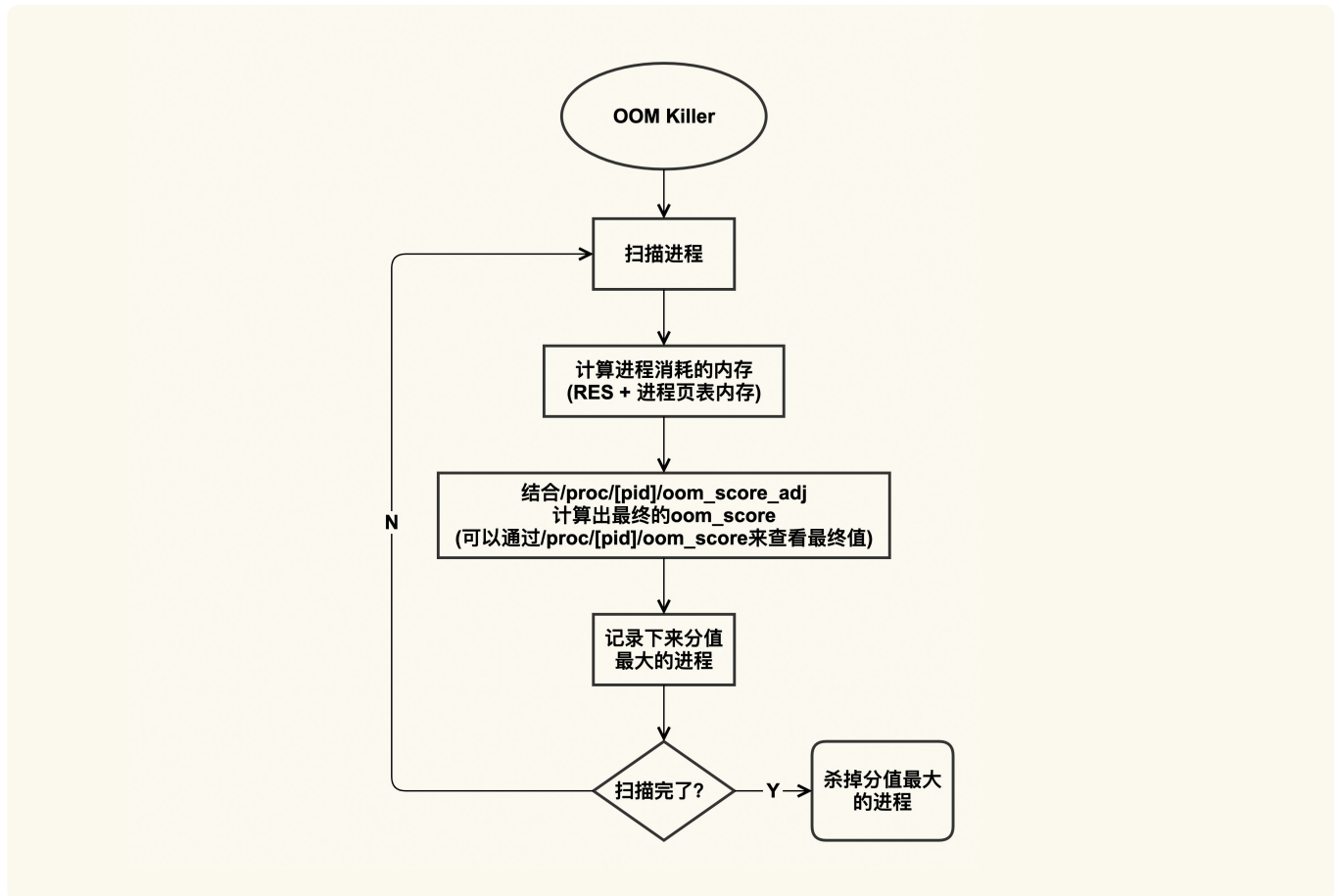
 复制代码

tmpfs 作为一种特殊的 Shmem, 它消耗的内存是不会体现在进程内存中的, 这往往会给问题排查带来一些难度。要想高效地分析这种类型的问题, 你必须要去熟悉系统中的内存类型。除了 tmpfs 之外, 其他一些类型的内存也不会体现在进程内存中, 比如内核消耗的内存: /proc/meminfo 中的 Slab (高速缓存)、KernelStack (内核栈) 和 VmallocUsed (内核通过 vmalloc 申请的内存), 这些也是你在不清楚内存被谁占用时要去排查的。

如果 tmpfs 消耗的内存越积越多而得不到清理，最终的结果也是系统可用内存不足，然后触发 OOM 来杀掉进程。它很有可能会杀掉很重要的进程，或者是那些你认为不应该被杀掉的进程。

## OOM 杀进程的危害

OOM 杀进程的逻辑大致如下图所示：



OOM killer 在杀进程的时候，会把系统中可以被杀掉的进程扫描一遍，根据进程占用的内存以及配置的 oom\_score\_adj 来计算出进程最终的得分，然后把得分 (oom\_score) 最大的进程给杀掉，如果得分最大的进程有多个，那就把先扫描到的那个给杀掉。

进程的 oom\_score 可以通过 /proc/[pid]/oom\_score 来查看，你可以扫描一下你系统中所有进程的 oom\_score，其中分值最大的那个就是在发生 OOM 时最先被杀掉的进程。不过你需要注意，由于 oom\_score 和进程的内存开销有关，而进程的内存开销又是会动态变化的，所以该值也会动态变化。

如果你不想这个进程被首先杀掉，那你可以调整该进程的 oom\_score\_adj 改变这个 oom\_score；如果你的进程无论如何都不能被杀掉，那你可以将 oom\_score\_adj 配置为



-1000。

通常而言，我们都需要将一些很重要的系统服务的 `oom_score_adj` 配置为 -1000，比如 `sshd`，因为这些系统服务一旦被杀掉，我们就很难再登陆进系统了。

但是，除了系统服务之外，不论你的业务程序有多重要，都尽量不要将它配置为 -1000。因为你的业务程序一旦发生了内存泄漏，而它又不能杀掉，这就会导致随着它的内存开销变大，OOM killer 不停地被唤醒，从而把其他进程一个个给杀掉，我们之前在生产环境中就遇到过类似的案例。

OOM killer 的作用之一，就是找到系统中不停泄漏内存的进程然后把它给杀掉，如果没有找对，那就会误杀其他进程，甚至是误杀了更为重要的业务进程。

OOM killer 除了会杀掉一些无辜进程外，它选择杀进程的策略也未必是正确的。接下来又到了给内核找茬的时刻了，这也是我们这个系列课程的目的：告诉你如何来学些 Linux 内核，但同时我也要告诉你，要对内核有怀疑态度。下面这个案例就是一个内核的 Bug：

在我们的一个服务器上，我们发现 OOM killer 在杀进程的时候，总是会杀掉最先扫描到的进程，而由于先扫描到的进程的内存太小，就导致 OOM 杀掉进程后很难释放出足够多的内存，然后很快再次发生 OOM。

这是在 Kubernetes 环境下触发的一个问题，Kubernetes 会将某些重要的容器配置为 `Guaranteed`（[对应的 `oom\_score\_adj` 为 -998](#)），以防止系统 OOM 的时候把该重要的容器给杀掉。然而，如果容器内部发生了 OOM 就会触发这个内核 Bug，导致总是杀掉最先扫描到的那个进程。

针对该内核 Bug，我也给社区贡献了一个 patch（[mm, oom: make the calculation of oom badness more accurate](#)）来修复这个选择不合适进程的问题，在这个 patch 的 commit log 里我详细地描述了该问题，感兴趣的话你可以去看下。

## 课堂总结

这节课，我们学习了 `tmpfs` 这种类型的内存泄漏以及它的观察方法，这种类型的内存泄漏和其他进程内存泄漏最大的不同是，你很难通过进程消耗的内存来判断是哪里在泄漏，因

为这种类型的内存不会体现在进程的 RES 中。但是, 如果你熟悉内存问题的常规分析方法, 你就能很快地找到问题所在。

在不清楚内存被谁消耗时, 你可以通过 `/proc/meminfo` 找到哪种类型的内存开销比较大, 然后再对这种类型的内存做针对性分析。

你需要配置合适的 OOM 策略 (`oom_score_adj`) 来防止重要的业务被过早杀掉 (比如将重要业务的 `oom_score_adj` 调小为负值), 同时你也需要考虑误杀其他进程, 你可以通过比较进程的 `/proc/[pid]/oom_score`, 来判断出进程被杀的先后顺序。

再次强调一遍, 你需要学习内核, 但同时你也需要对内核持怀疑态度。

总之, 你对不同内存类型的特点了解越多, 你在分析内存问题的时候 (比如内存泄漏问题) 就会更加高效。熟练掌握这些不同的内存类型, 你也能够在业务需要申请内存时选择合适的内存类型。

## 课后作业

请你运行几个程序, 分别设置不同的 `oom_score_adj`, 并记录下它们的 `oom_score` 是什么样的, 然后消耗系统内存触发 OOM, 看看 `oom_score` 和进程被杀的顺序是什么关系。欢迎你在留言区与我讨论。

感谢你的阅读, 如果你认为这节课的内容有收获, 也欢迎把它分享给你的朋友, 我们下一讲见。

提建议

## 更多课程推荐

## 程序员的数学基础课

在实战中重新理解数学

黄申

LinkedIn 资深数据科学家



涨价倒计时 🕒

今日秒杀 **¥79**, 9月11日涨价至 **¥129**

© 版权归极客邦科技所有, 未经许可不得传播售卖。页面已增加防盗追踪, 如有侵权极客邦将依法追究其法律责任。

上一篇 07 案例篇 | 如何预防内存泄漏导致的系统假死?

下一篇 09 分析篇 | 如何对内核内存泄漏做些基础的分析?

## 精选留言 (9)

💬 写留言



Linuxer

2020-09-05

赞! 意犹未尽, 爽!

展开 ▾



2



一千零一夜

2020-09-05

/PROC/MEMINFO之谜: <http://linuxperf.com/?p=142>

1



欧阳

2020-09-10



老师好, 看完这一讲有两个疑问想请教:

(1) tmpfs的特点是快, 那么与内存有什么不同呢?

假设一个app只有白天可访问, 晚上不提供服务。

白天用tmpfs, 晚上再来做耗时的部分: 写磁盘, 清理tmpfs。

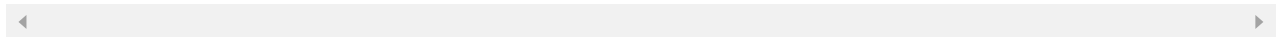
那这样的话, 为什么白天不直接存储在内存就好, 不必写入tmpfs吧? ...

展开 ∨

作者回复: 1. tmpfs是内存的一种使用形式, tmpfs里的内容都在内存中。

文件内容在内存中可能会被回收掉, 而tmpfs不用担心这个问题; 如果你不想被回收, 那就需要mlock它。所以tmpfs的优势是使用方便。

2. mmap是接口, tmpfs是存储方式。二者位于不同的层, mmap也可以使用tmpfs, 如果进程mmap方式打开tmpfs文件还没有unmap它, 那这部分tmpfs内存是属于进程地址空间的。

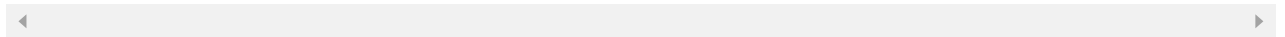


qinsi

2020-09-09

看了下那个patch, 为啥原始代码里要把负分给截断阿, 是有什么特别的考虑吗

作者回复: 没有特别的考虑, 只是因为没有考虑到会存在大量进程设置为负分的这种场景。



KennyQ

2020-09-06

有几个问题:

1. 哪些进程会去使用tmpfs?
2. 如何使用tmpfs, 是在启动进程的时候把代码嵌入进去? 还是说有接口可以调用?
3. tmpfs是怎么被挂载起来的? 重启后是否会自动挂载?
4. tmpfs如果被100个进程调用了, 会挂载100次么? ...

展开 ∨

作者回复: 1. 很多应用程序会把日志保存在tmpfs里, 也有些应用会把它的运行时文件保存在tmpfs里。比如systemd就会把日志记录到tmpfs中。你可以用mount命令来查看你的系统tmpfs挂载路径, 然后去这里面查看有哪些文件, 你也可以看出来你的系统里有哪些应用会用到tmpfs。

2. 最简单的使用方式是把tmpfs给mount到一个路径上 然后你的应用往里面写文件就可以使用它了。

3. 通过mount命令来挂载, 为了重启后也生效, 你可以在启动脚本里来挂载。

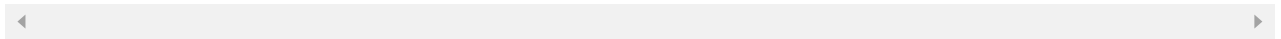
4. 不会, 只需要挂载一次 然后所有进程都可以用了 使用方式上跟普通文件路径没有差异。

5. 会的。

6. tmpfs的目的是针对那些读写io频繁, 但数据量不大的场景。既避免了io, 又不浪费太多内存。

如果你的业务有这种场景, 你可以考虑。

7. 可以的 remount可以改变它的大小。



**老酒馆**

2020-09-06

精彩

展开 ∨



**ray**

2020-09-06

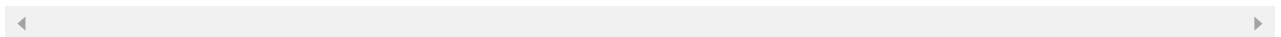
老师您好,

请问我们该怎么判断一个process会不会使用到tmpfs呢?

谢谢老师的解答^^

展开 ∨

作者回复: 如果进程打开的tmpfs文件还没有close, 那么从进程打开的fd里就能看到这些内存文件(比如通过lsfd); 如果还没有unmap()掉它从tmpfs里申请的内存, 那么从进程的地址空间里是可以观察到这些tmpfs内存的。而如果进程已经close掉了它打开的tmpfs文件, 那就无法通过进程来分析了, 但是通常tmpfs文件里的文件名都会加上进程的一些相关标记, 以方便观察是哪个应用创建的。所以我的建议是, 你在使用tmpfs文件时, 最好这些文件名能够跟你的业务有些关联。

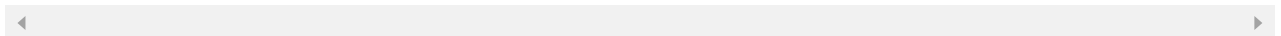


**jssfy**

2020-09-05

这个patch很赞! 请问老师这个mm oom的bug在centos 7下是否也有?

作者回复: 对的 这个bug在centos7上也存在 可以backport到centos7上。



**飞翔**



2020-09-05

我这里有一个这样的问题，服务器集群每隔一段时间就有机器进入死机状态，可以ping通，但是ssh连不上，这时候就得联系现场人员去重启机器，很麻烦。怀疑是内存耗尽的原因，但是重启机器后，从message日志又看不到证据，oom是不是不一定会释放内存？这种问题应该怎么定位呢？有什么解决方案呢？

展开 ✓

作者回复: 你的内核版本是什么？在老版本内核里，oom如果杀的进程阻塞在内核态的话，比如处于D状态，那这个进程的anon内存就没有办法释放；如果是新版本的话，oom会唤醒oom\_reaper 然后这个reaper会释放阻塞的进程的内存。所以这类问题的解决方案是，backport oom reaper机制，或者升级内核。定位手段是，你可以使用sysrq -w来看是否这个被杀的进程处于d状态。



2

