

**Linux内存统计及泄漏问题定位**

（仅供内部使用）

|  |  |  |
| --- | --- | --- |
| 版 本 号： | | V0.1 |
|  | | |
| 编 制： | 吴杰 | |
| 审 核： |  | |

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| *2016/03/30* | *0.1* | *初稿完成* | *吴杰* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 案例描述 2](#_Toc183256821)

[2 案例分析 2](#_Toc183256822)

[3 解决过程 2](#_Toc183256823)

[4 解决结果 2](#_Toc183256824)

[5 总结 2](#_Toc183256825)

**关键词：**

内存统计，内存泄漏

**摘 要：**

介绍了linux中内存统计相关的一些知识，以及内存泄漏问题如何定位。

# 案例描述

介绍了linux中内存统计相关的一些知识，以及内存泄漏问题如何定位。

# 案例分析

在产品的开发中，通过对当前系统消耗内存总量的统计，可以对产品所需内存总量进行精确的评估，从而选择合适的内存芯片与大小，降低产品的成本。在遇到内存泄露类问题时，经常会对此束手无策，本文通过对proc下进程相关的文件进行分析，精确评估系统消耗内存的大小，还可以对内存泄露类问题的解决提供一种定位手段。

# 解决过程

Linux在内存使用上的原则是：如果内存充足，不用白不用，尽量使用内存来缓存一些文件，从而加快进程的运行速度，而当内存不足时，会通过相应的内存回收策略收回cache内存，供进程使用。

一、系统总内存的分析。

可以从proc目录下的meminfo文件了解到当前系统内存的使用情况汇总，其中可用的物理内存=memfree+buffers+cached，当memfree不够时，内核会通过回写机制(pdflush线程)把cached和buffered内存回写到后备存储器，从而释放相关内存供进程使用，或者通过手动方式显式释放cache内存

   echo 3 > /proc/sys/vm/drop\_caches

下图是海思平台下当前系统内存的总体使用情况，其中可以看到，系统消耗掉了29M的内存，下面继续分析这些内存都是被谁消耗掉了。

# cat /proc/meminfo

MemTotal:        68956 kB

MemFree:         18632 kB

Buffers:          4096 kB

Cached:          17260 kB

SwapCached:          0 kB

Active:          21304 kB

Inactive:        19248 kB

SwapTotal:           0 kB

SwapFree:            0 kB

Dirty:               0 kB

Writeback:           0 kB

AnonPages:       19216 kB

Mapped:           2472 kB

Slab:             6900 kB

SReclaimable:      924 kB

SUnreclaim:       5976 kB

PageTables:        460 kB

NFS\_Unstable:        0 kB

Bounce:              0 kB

CommitLimit:     62060 kB

Committed\_AS:    28864 kB

VmallocTotal:   442368 kB

VmallocUsed:     46984 kB

VmallocChunk:   393212 kB

二、进程使用内存的统计

在32位操作系统中，每个进程拥有4G的虚拟内存空间，其中0~3GB是每个进程的私有用户空间，这个空间对系统中其他进程是不可见的。3~4GB是linux内核空间，由系统所有的进程以及内核所共享的。通过访问/proc/{pid}/下相关文件，可以了解每个线程虚拟内存空间的使用情况，从而了解每个线程所消耗内存的多少。

由于我们的产品都是使用多线程方式实现的，多个线程共享一个进程的用户态虚拟地址空间，虚拟地址空间包含若干区域，主要有如下几个区域：

1、当前执行文件的代码段，该代码段称为text段。

2、执行文件的数据段，主要存储执行文件用到的全局变量，静态变量。

3、存储全局变量和动态产生的数据的堆。

4、用于保存局部变量和实现函数调用的栈。

5、采用mmap方式映射到虚拟地址空间中的内存段

所以只需要查看任意一个线程的用户态虚拟地址空间分配即可知道属于同一进程的所有线程占用总内存的大小。可以通过查看/proc/{pid}/maps文件来获取相关的虚拟地址空间内容，下文摘列部分典型的内容：

# cat /proc/568/maps

00008000-0036a000 r-xp 00000000 00:0e 236        /home/hik/hicore

00372000-003a5000 rw-p 00362000 00:0e 236        /home/hik/hicore

003a5000-00e28000 rwxp 003a5000 00:00 0          [heap]

40000000-40005000 r-xp 00000000 01:00 94         /lib/ld-uClibc.so.0

416db000-41770000 rw-s c2005000 00:0f 68         /dev/mem

b51fc000-b5200000 rwxp b51fc000 00:00 0

…….

be1fc000-be200000 rwxp be1fc000 00:00 0

be93b000-be950000 rwxp befeb000 00:00 0          [stack]

第一行：从r-xp可知其权限为只读、可执行，该段内存地址对应于执行文件的

代码段，程序的代码段需加载到内存中才可以执行。由于其只读，不会

被修改，所以在整个系统内共享。

第二行：从rw-p可知其权限为可读写，不可执行，该段内存地址对应于执行文件的数据段，存放执行文件所用到的全局变量、静态变量。

第三行：从rwxp可知其权限是可读写，可执行，地址空间向上增长，而且不对应文件，是堆段，进程使用malloc申请的内存放在堆段。每个进程只有一个堆段，不论是主进程，还是不同的线程申请的内存，都反映到到进程的堆段。堆段向上增长，最大可以增长到1GB的位置，即0x40000000，如果大于1GB，glibc将采用mmap的方式，为堆申请一块内存。

第四行：是程序连接的共享库的内存地址。

第五行：是以mmap方式映射的虚拟地址空间。

第六、七行：是线程的栈区地址段，每个线程的栈大小都是16K。

第八行：是进程的栈区。关于栈段，每个线程都有一个，如果进程中有多个线程，则包含多个栈段。

三、当前系统总内存的统计

    1、进程占用的总内存可以通过上述maps表计算出来。

    2、当系统运行起来以后，会把应用层相关的文件挂载到tmpfs文件系统下，海思系统下这部分大概有13M左右，这部分内存是以cache方式统计出来的，但是这部分内存cache无法通过回收策略或者显式的调用释放掉。

    3、根文件系统ramdisk占用的内存。

    4、当前系统保留内存的大小，可以通过查看/proc/sys/vm/min\_free\_kbytes来获取或者修改此内存的大小。

    5、当然，当系统运行起来后，还应该留有一定的内存用于在硬盘读写时做cache或者网络负荷比较高时分配skb等，一般需要30M以上。

四、对调试内存泄露类问题的一些启示

   当进程申请内存时，实际上是glibc中内置的内存管理器接收了该请求，随着进程申请内存的增加，内存管理器会通过系统调用陷入内核，从而为进程分配更多的内存。

针对堆段的管理，内核提供了两个系统调用brk和mmap，brk用于更改堆顶地址，而mmap则为进程分配一块虚拟地址空间。

当进程向glibc申请内存时，如果申请内存的数量大于一个阀值的时候，glibc会采用mmap为进程分配一块虚拟地址空间，而不是采用brk来扩展堆顶的指针。缺省情况下，此阀值是128K，可以通过函数来修改此值。

             Int mallopt(int param, int value)

Param的取值分别为M\_MMAP\_THRESHOLD、M\_MMAP\_MAX。

Value的取值是以字节为单位的。

M\_MMAP\_THRESHOLD是glibc中申请大块内存阀值，大于该阀值的内存申请，内存管理器将使用mmap系统调用申请内存，如果小于该阀值的内存申请，内存管理器使用brk系统调用扩展堆顶指针。

M\_MMAP\_MAX是该进程中最多使用mmap分配地址段的数量。

# 解决结果

如果在实际的调试过程中，怀疑某处发生了内存泄露，可以查看该进程的maps表，看进程的堆段或者mmap段的虚拟地址空间是否持续增加，如果是，说明很可能发生了内存泄露，如果mmap段虚拟地址空间持续增加，还可以看到各个段的虚拟地址空间的大小，从而可以确定是申请了多大的内存，对调试内存泄露类问题可以起到很好的定位作用。

# 总结

无