kmemleak查看内核态内存泄露

一、介绍

kmemleak主要用来检测内存泄露，检测对象

kmalloc/kzalloc

vmalloc

kmem\_cache\_alloc等分配的内存块

二、用法

1.编译内核前，使能CONFIG\_DEBUG\_KMEMLEAK

在初始化kmemleak之前，内存的分配或释放这些动作被存储在一个前期日志缓冲区。这个缓冲区的大小通过配CONFIG\_DEBUG\_KMEMLEAK\_EARLY\_LOG\_SIZE设置。（默认大小是400，如果采用默认大小，后续挂载文件系统，可能找不到kmemleak，查看dmesg，log\_size太小，kmemleak开启失败。）

2.挂载debugfs文件系统

mount -t debugfs nodev /sys/kernel/debug/

3.开启内核自动检测线程

echo scan > /sys/kernel/debug/kmemleak

4.查看打印信息

cat /sys/kernel/debug/kmemleak

5.清除内核检测报告，新的内存泄露报告将重新写入/sys/kernel/debug/kmemleak

echo clear > /sys/kernel/debug/kmemleak

6.其他参数说明

|  |  |
| --- | --- |
| 参数 | 内容 |
| off | 禁用kmemleak。不再跟踪内存分配和释放。一旦禁用，就不能再次开启。 |
| stack=on | 启用线程栈区域的扫描。默认是on。 |
| stack=off | 禁用线程栈区域的扫描。 |
| scan=on | 开启kmemleak内核线程的自动扫描。默认为on。 |
| scan=off | 停止kmemleak内核线程的自动扫描。 |
| scan= | 设置kmemleak线程执行扫描时间间隔。单位为秒，默认为600s(10分钟)。0s表示停止自动扫描。 |
| scan | 手动触发扫描，立即扫描。 |
| clear | 清除检测出的数据，即清除之前判断为内存泄露的object信息，会把这些object标记为KMEMLEAK\_GREY，并不会把object从红黑树和双向链表中删除，不在/sys/kernel/debug/kmemleak中显示，只是不显示，可以使用dump参数进行确定。在使用kmemleak前清除没有关系的信息时使用。 |
| dump= | 显示addr对应object信息。 |

三、使用示例

1.设置CONFIG\_DEBUG\_KMEMLEAK\_TEST以模块形式编译，文件在mm/kmemleak-test.c，帮助文档在Documentation/kmemleak.txt

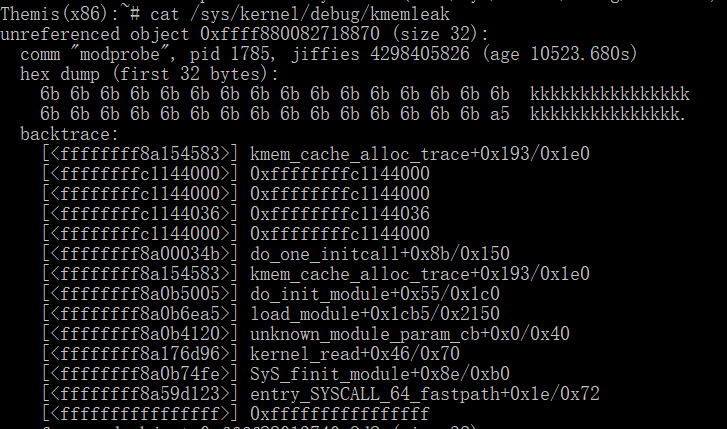
2.加载模块，并执行扫描

# modprobe kmemleak-test

# echo scan > /sys/kernel/debug/kmemleak

3.cat /sys/kernel/debug/kmemleak查看疑似内存泄露（该工具小概率存在漏报和误报）

# cat /sys/kernel/debug/kmemleak



分析：

1、产生泄漏内存块对应的object的起始地址为0xffff880082718870，

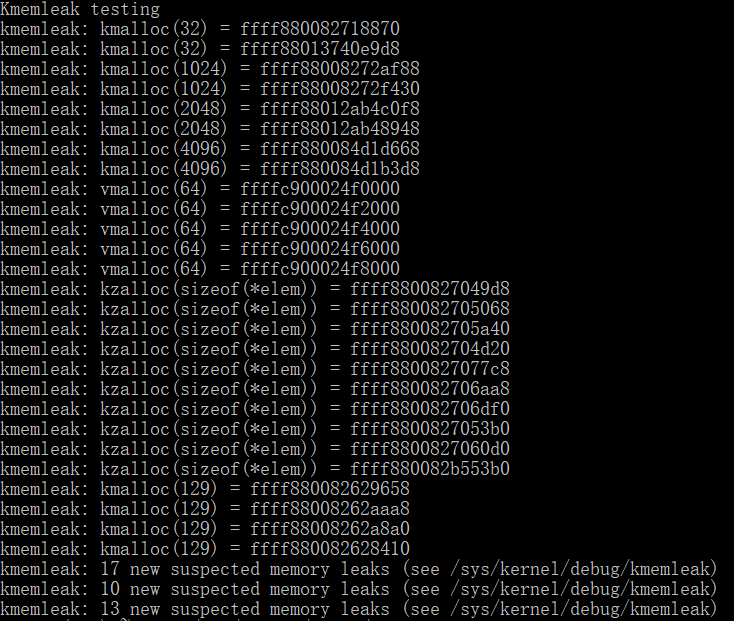
泄漏的大小为32个字节。

2、进程名为modprobe，pid为1785，创建object时的jiffies为10523.680s。

3、泄漏内存块的前32字节数据。

4、泄漏点的backtrace信息。

4.Dmesg查看代码里加的指针打印信息，对照kmemleak可以看出是哪个指针没有释放



四、原理

具体实现在mm/kmemleak.c中

kmemleak(kernel memory leak detector)是检测内核空间的内存泄漏的调试工具。检测对象是memblock\_alloc、kmalloc、vmalloc、kmem\_cache\_alloc等函数分配的内存块，该内存块由struct kmemleak\_object来描述(简称为object)。kmemleak的实现原理非常简单，通过暴力扫描内存（假定内存中存放的都是指针，以ARM64为例，每次扫描8个字节)，如果找不到指向起始地址或者内存块任何位置的指针，则分配的内存块被认为是孤立的。这意味着内核可能无法将分配内存块的地址传递给释放函数，因此该内存块被视为内存泄漏。内存块(object)有3种颜色，分别为黑色、白色、灰色， 通过count和min\_count区分不同颜色的object。

黑色: min\_count = -1，表示被忽略的object，此object不包含对别人的引用，也不会存在内存泄漏，比如代码段会标记为黑色。

白色: count < min\_count，孤立的object，没有足够的引用指向这个object，一轮扫描结束后被认为泄漏的内存块。

灰色: min\_count = 0，表示不是孤立的object，即不存在内存泄漏的object，如代码中主动标记object为灰色，防止误报（如data、bss、ro\_after\_init）。或者count >= min\_count，对该object有足够的指针引用，认为不存在内存泄漏的内存块。

具体检测步骤如下：

1、通过struct kmemleak\_object(简称为object)描述kmalloc、vmalloc、kmem\_cache\_alloc等函数申请的内存块，记录申请内存的起始地址，大小、call trace等信息。同时把object加入到红黑树object\_tree\_root和双向链表object\_list中，红黑树中的key值为内存块的起始地址。

2、遍历双向链表object\_list，把所有的object的count计数清0，即在新的一轮扫描前，尽可能的把能复位成白色的object标记为白色。然后判断object是否是灰色(默认data、bss、ro\_after\_init段会被标记为灰色)，如果是灰色的object则把object加入到灰色链表gray\_list中。

3、扫描内存中可能存放指针的内存区域(per-cpu段、struct page的内容、内核栈、灰色链表),根据挂在红黑树中所有的object的地址范围进行对比。如果有指针指向某一个object(指向该object的起始地址或者指向object地址范围内)，会把object对应的count字段增加1，如果object变成灰色，则会把object加入到灰色链表中。

4、扫描object\_list中的白色对象的object，判断object所描述的地址范围的内容的crc值是否发生变化，如果发生变化，则同样把object加入到灰色链表gray\_list中。说明通过间接的方式访问了object描述的地址范围，不是内存泄漏，减少误报。

5、重新扫描灰色链表，因为步骤4中，可能有些白色的object加入到了灰色链表中，需要重新扫描。

6、经过上述一系列的扫描，剩余白色的object就是可疑的内存泄漏点。

主要参考：

1. Documentation/kmemleak.txt

2. https://blog.csdn.net/lishenglong666/article/details/8287783

3. http://news.eeworld.com.cn/mp/ymc/a132665.jspx