Linux任督二脉之内存管理(二)

麦当劳喜欢您来，喜欢您再来

内存的动态申请和释放

* slab、kmalloc/kfree、/proc/slabinfo和slabtop
* 用户空间malloc/free与内核之间的关系
* mallopt
* vmalloc
* 内存耗尽（OOM）、oom_score和oom_adj
* Android进程生命周期与OOM

练习题

*看/proc/slabinfo，运行slabtop
*运行mallopt.c程序：mallopt(M_TRIM_THRESHOLD)等
*看/proc/vmallocinfo,grep ioremap映射
*运行一个很耗费内存的程序，观察oom memory
*通过oom_adj调整firefox的oom_score

Buddy的问题

分配的粒度太大

1页

2页

4页...

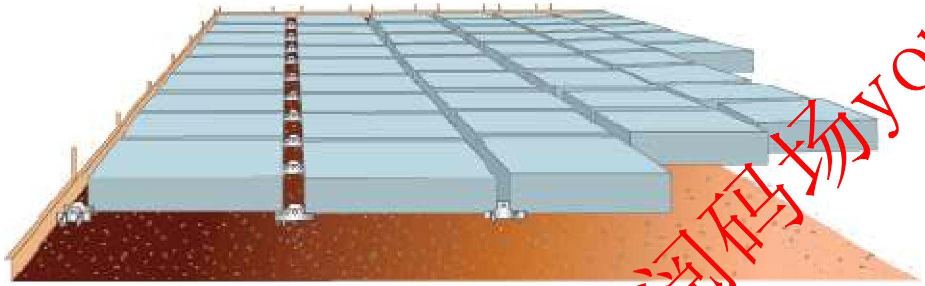
我们常常需要分配小内存

能否先分大的，再划分小的等份？

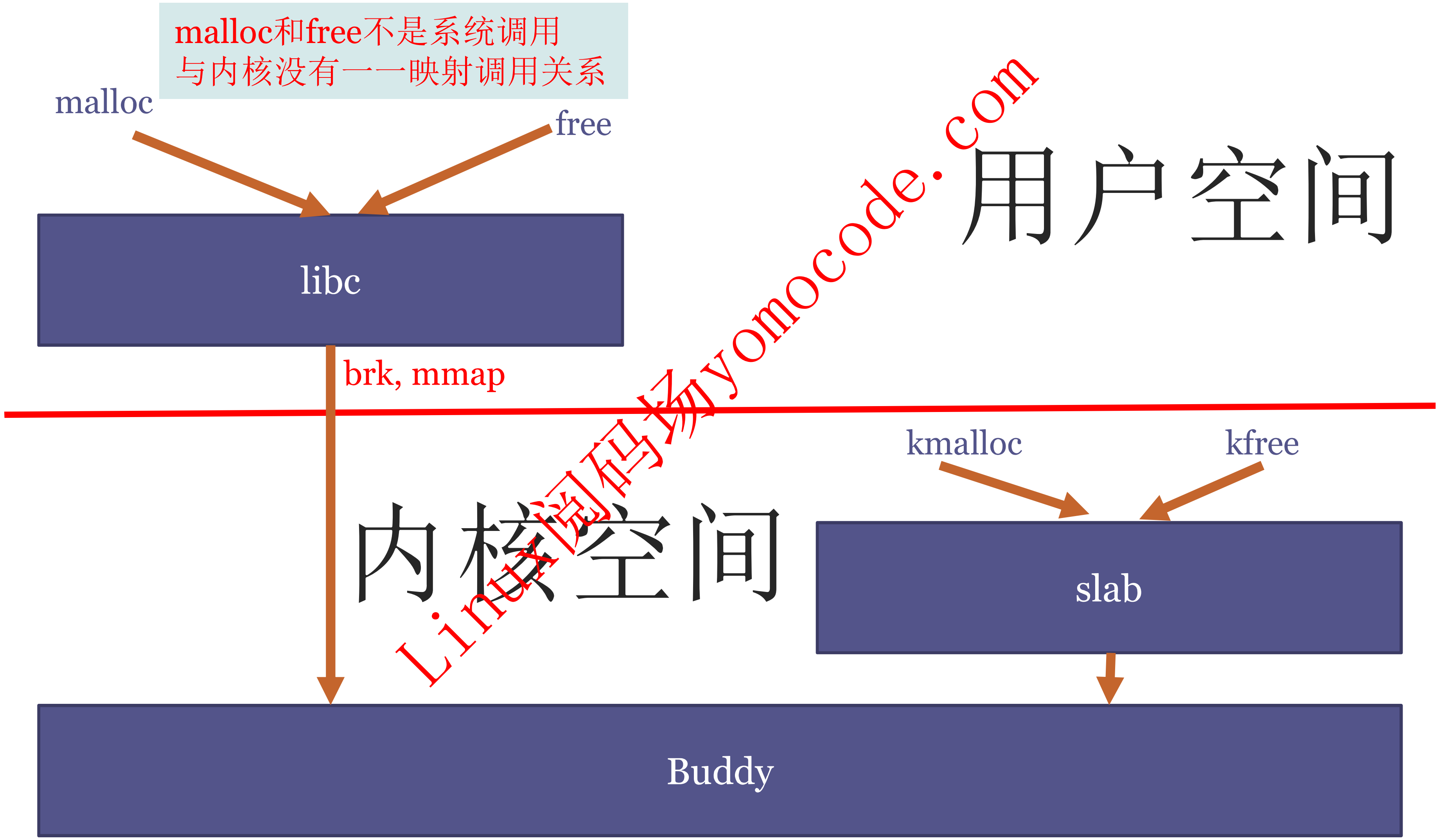
Slab

slab
n. 厚板，平板，厚片; 混凝土路面;
v. 把…分成厚片; 用石板铺;

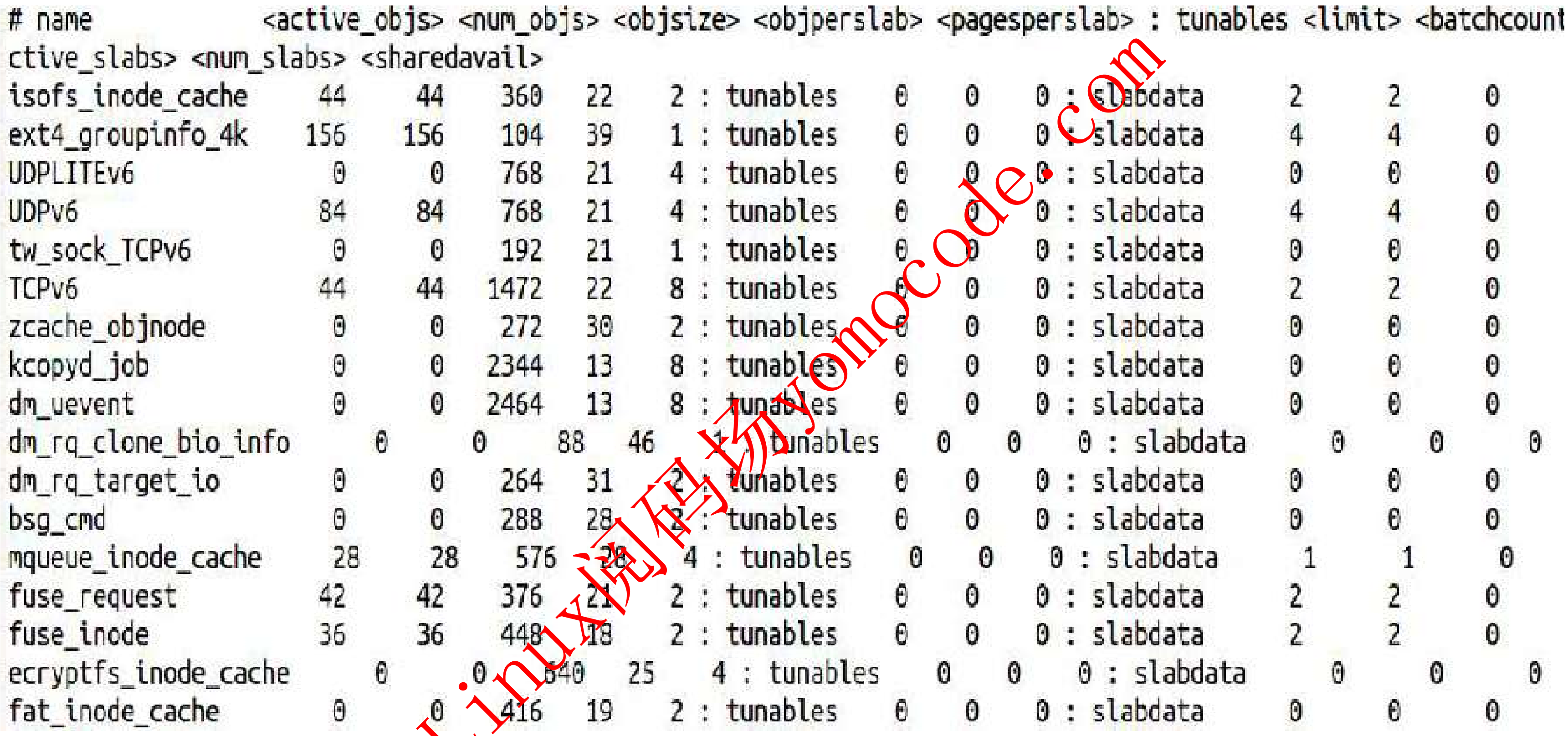
View of a waffle raft slab being set up, showing the cardboard void forms



Libc，Slab与buddy

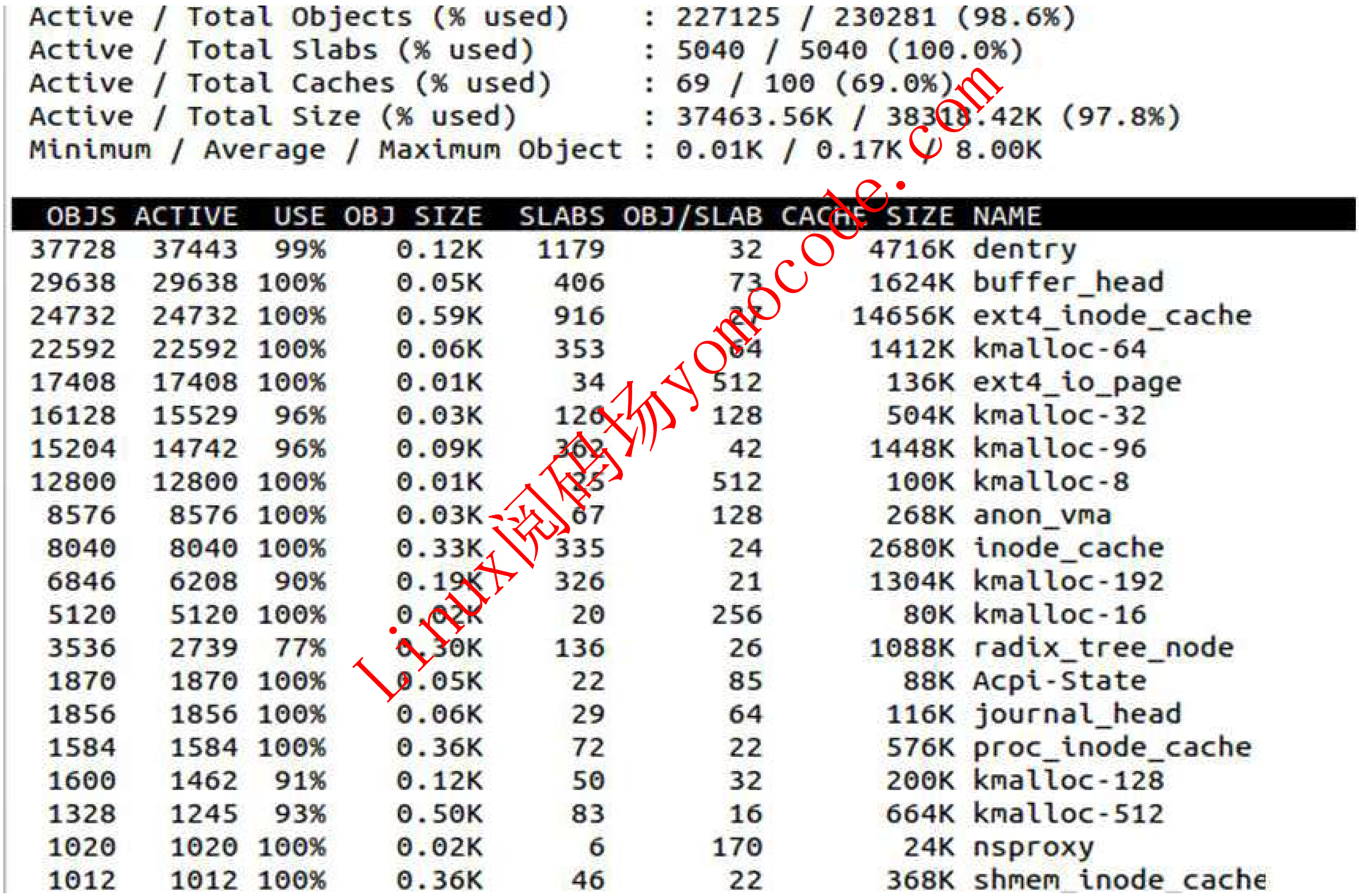

malloc和free不是系统调用
与内核没有一一映射调用关系

/proc/slabinfo



```
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount
ctive_slabs> <num_slabs> <sharedavail>
isofs_inode_cache      44     44    360   22    2 : tunables    0    0    0 : slabdata      2      2      0
ext4_groupinfo_4k     156    156    104   39    1 : tunables    0    0    0 : slabdata      4      4      0
UDPLITEv6               0      0    768   21    4 : tunables    0    0    0 : slabdata      0      0      0
UDPv6                  84     84    768   21    4 : tunables    0    0    0 : slabdata      4      4      0
tw_sock_TCPv6           0      0    192   21    1 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                  44     44   1472   22    8 : tunables    0    0    0 : slabdata      2      2      0
zcache_objnode          0      0    272   30    2 : tunables    0    0    0 : slabdata      0      0      0
kcopyd_job              0      0   2344   13    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent               0      0   2464   13    8 : tunables    0    0    0 : slabdata      0      0      0
dm_rq_clone_bio_info      0      0     88   46    1 : tunables    0    0    0 : slabdata      0      0      0
dm_rq_target_io         0      0    264   31    2 : tunables    0    0    0 : slabdata      0      0      0
bsg_cmd                 0      0    288   28    2 : tunables    0    0    0 : slabdata      0      0      0
mqueue_inode_cache     28     28    576   28    4 : tunables    0    0    0 : slabdata      1      1      0
fuse_request           42     42    376   21    2 : tunables    0    0    0 : slabdata      2      2      0
fuse_inode             36     36    448   18    2 : tunables    0    0    0 : slabdata      2      2      0
ecryptfs_inode_cache      0      0    640   25    4 : tunables    0    0    0 : slabdata      0      0      0
fat_inode_cache         0      0    416   19    2 : tunables    0    0    0 : slabdata      0      0      0
```

slabtop



```
 Active / Total Objects (% used)    : 227125 / 230281 (98.6%)
 Active / Total Slabs (% used)      : 5040 / 5040 (100.0%)
 Active / Total Caches (% used)     : 69 / 100 (69.0%)
 Active / Total Size (% used)       : 37463.56K / 38318.42K (97.8%)
 Minimum / Average / Maximum Object : 0.01K / 0.17K / 8.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
 37728  37443  99%    0.12K   1179       32      4716K dentry
 29638  29638 100%    0.05K    406       73      1624K buffer_head
 24732  24732 100%    0.59K    916       27     14656K ext4_inode_cache
 22592  22592 100%    0.06K    353       64      1412K kmalloc-64
 17408  17408 100%    0.01K     34      512       136K ext4_io_page
 16128  15529  96%    0.03K    126      128       504K kmalloc-32
 15204  14742  96%    0.09K    362       42      1448K kmalloc-96
 12800  12800 100%    0.01K     25      512       100K kmalloc-8
  8576   8576 100%    0.03K     67      128       268K anon_vma
  8040   8040 100%    0.33K    335       24      2680K inode_cache
  6846   6208  90%    0.19K    326       21      1304K kmalloc-192
  5120   5120 100%    0.02K     20      256        80K kmalloc-16
  3536   2739  77%    0.30K    136       26      1088K radix_tree_node
  1870   1870 100%    0.05K     22       85        88K Acpi-State
  1856   1856 100%    0.06K     29       64       116K journal_head
  1584   1584 100%    0.36K     72       22       576K proc_inode_cache
  1600   1462  91%    0.12K     50       32       200K kmalloc-128
  1328   1245  93%    0.50K     83       16       664K kmalloc-512
  1020   1020 100%    0.02K      6      170        24K nsproxy
  1012   1012 100%    0.36K     46       22       368K shmem_inode_cache
```

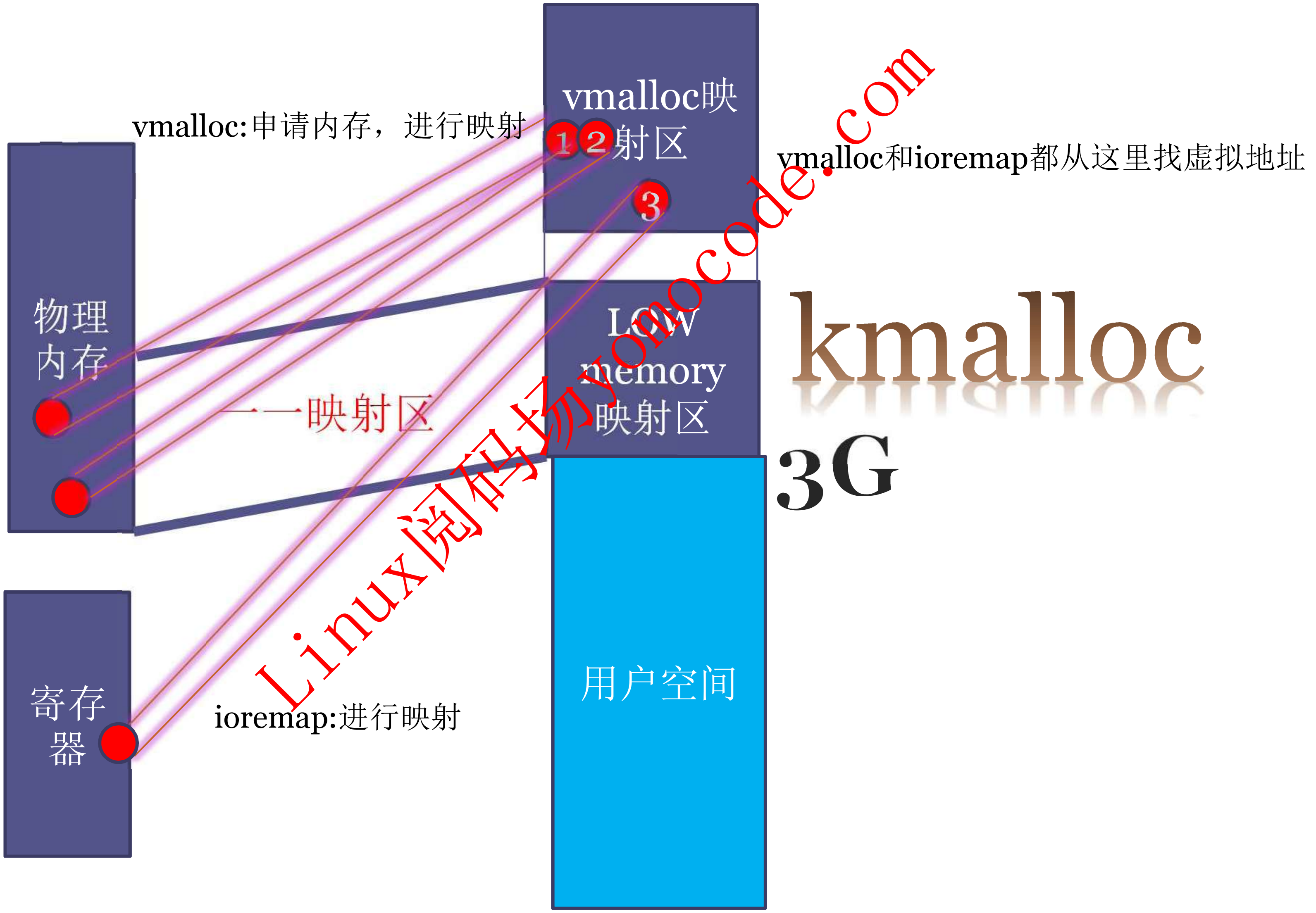
mallopt

```
#include <malloc.h>
#include <sys/mman.h>

#define SOMESIZE (100*1024*1024)     // 100MB

int main(int argc, char *argv[])
{
        unsigned char *buffer;
        int i;

        if (!mlockall(MCL_CURRENT | MCL_FUTURE))
                mallopt(M_TRIM_THRESHOLD, -1UL);
        mallopt(M_MMAP_MAX, 0);

        buffer = malloc(SOMESIZE);
        if (!buffer)
                exit(-1);

        /*
         * Touch each page in this piece of memory to get it
         * mapped into RAM
         */
        for (i = 0; i < SOMESIZE; i += 4 * 1024)
                buffer[i] = 0;
        free(buffer);
        /* <do your RT-thing> */

        while(1);
        return 0;
}
```

设置收缩阈值

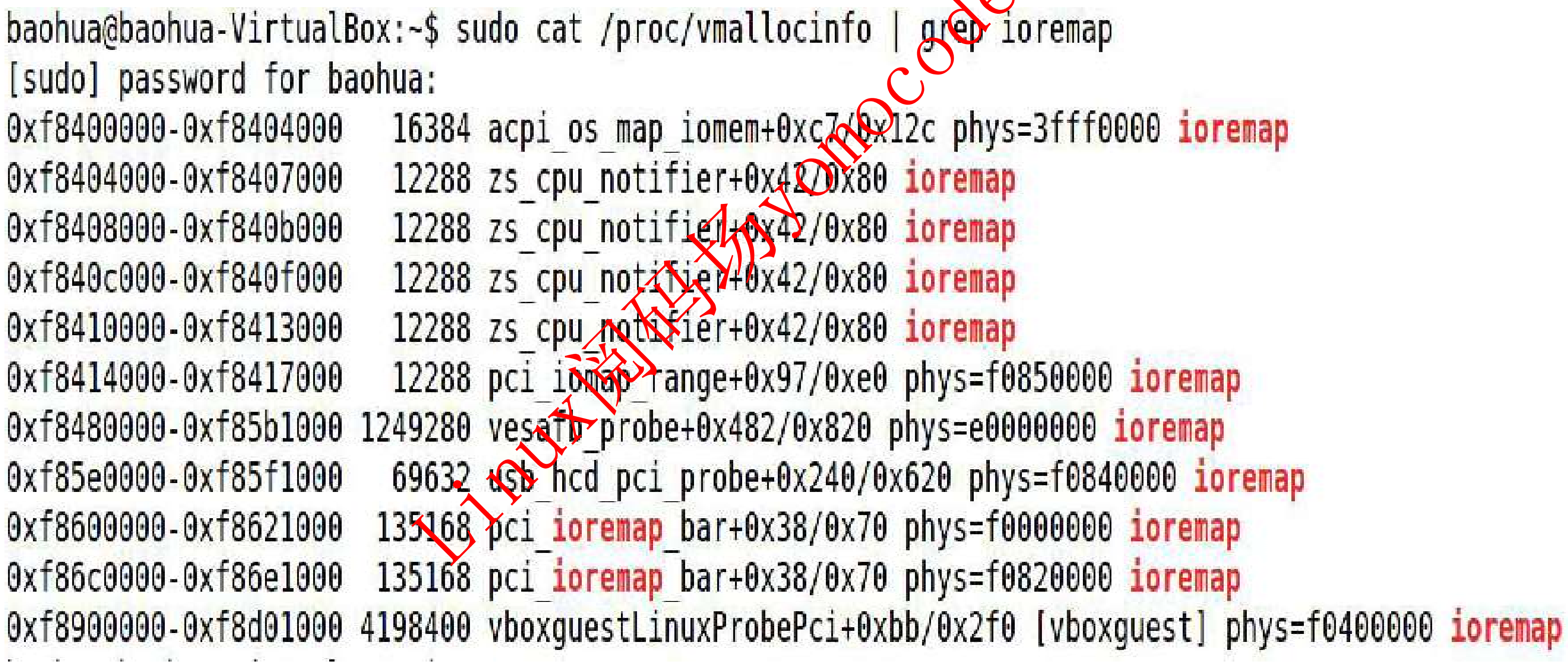
此时内存不还给内核

kmalloc vs. vmalloc/ioremap



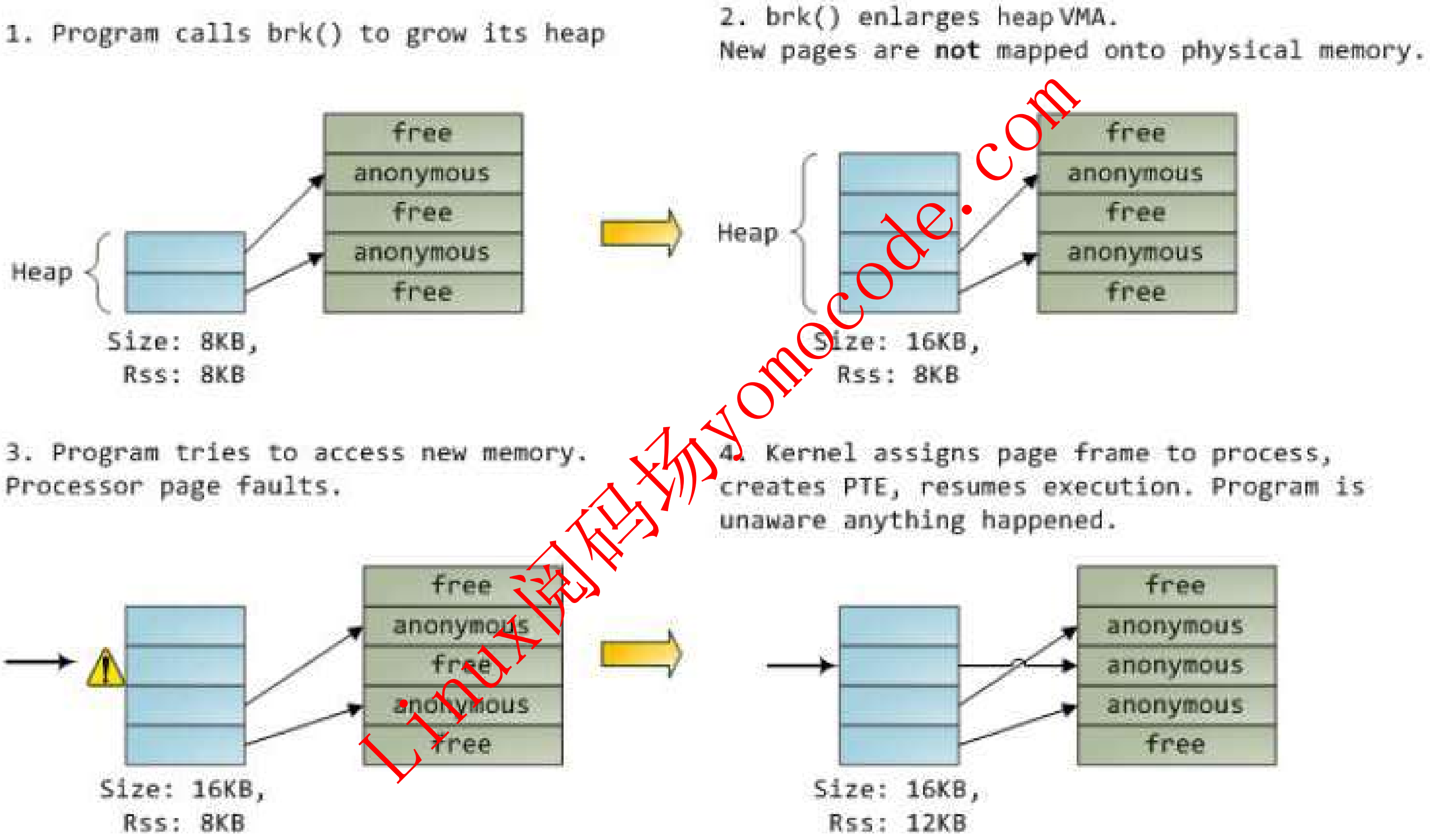
vmallocinfo

从这里可以看出寄存器映射的情况



```
baohua@baohua-VirtualBox:~$ sudo cat /proc/vmallocinfo | grep ioremap
[sudo] password for baohua:
0xf8400000-0xf8404000   16384 acpi_os_map_iomem+0xc7/0x12c phys=3fff0000 ioremap
0xf8404000-0xf8407000   12288 zs_cpu_notifier+0x42/0x80 ioremap
0xf8408000-0xf840b000   12288 zs_cpu_notifier+0x42/0x80 ioremap
0xf840c000-0xf840f000   12288 zs_cpu_notifier+0x42/0x80 ioremap
0xf8410000-0xf8413000   12288 zs_cpu_notifier+0x42/0x80 ioremap
0xf8414000-0xf8417000   12288 pci_iomap_range+0x97/0xe0 phys=f0850000 ioremap
0xf8480000-0xf85b1000 1249280 vesafb_probe+0x482/0x820 phys=e0000000 ioremap
0xf85e0000-0xf85f1000   69632 usb_hcd_pci_probe+0x240/0x620 phys=f0840000 ioremap
0xf8600000-0xf8621000  135168 pci_ioremap_bar+0x38/0x70 phys=f0000000 ioremap
0xf86c0000-0xf86e1000  135168 pci_ioremap_bar+0x38/0x70 phys=f0820000 ioremap
0xf8900000-0xf8d01000 4198400 vboxguestLinuxProbePci+0xbb/0x2f0 [vboxguest] phys=f0400000 ioremap
```

malloc: VSS vs. RSS



1. Program calls brk() to grow its heap

2. brk() enlarges heap VMA. New pages are **not** mapped onto physical memory.

3. Program tries to access new memory. Processor page faults.

4. Kernel assigns page frame to process, creates PTE, resumes execution. Program is unaware anything happened.

内存耗尽的程序

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
        int max = -1;
        int mb = 0;
        char *buffer;
        int i;
#define SIZE 2000
        unsigned int *p = malloc(1024 * 1024 * SIZE);

        printf("malloc buffer: %p\n", p);

        for (i = 0; i < 1024 * 1024 * (SIZE/sizeof(int)); i++) {
                p[i] = 123;
                if ((i & 0xFFFFF) == 0) {
                        printf("%dMB written\n", i >> 18);
                        usleep(100000);
                }
        }
        pause();
        return 0;
}
```

设定条件

- ✓ 总内存1G
- ✓ swapoff –a
- ✓ echo 1 > /proc/sys/vm/overcommit_memory

写到后面，会OOM

OOM打分因子

- mm/oom_kill.c中的badness() 给每个进程一个oom score，取决于：

✓ 驻留内存、 pagetable和swap的使用

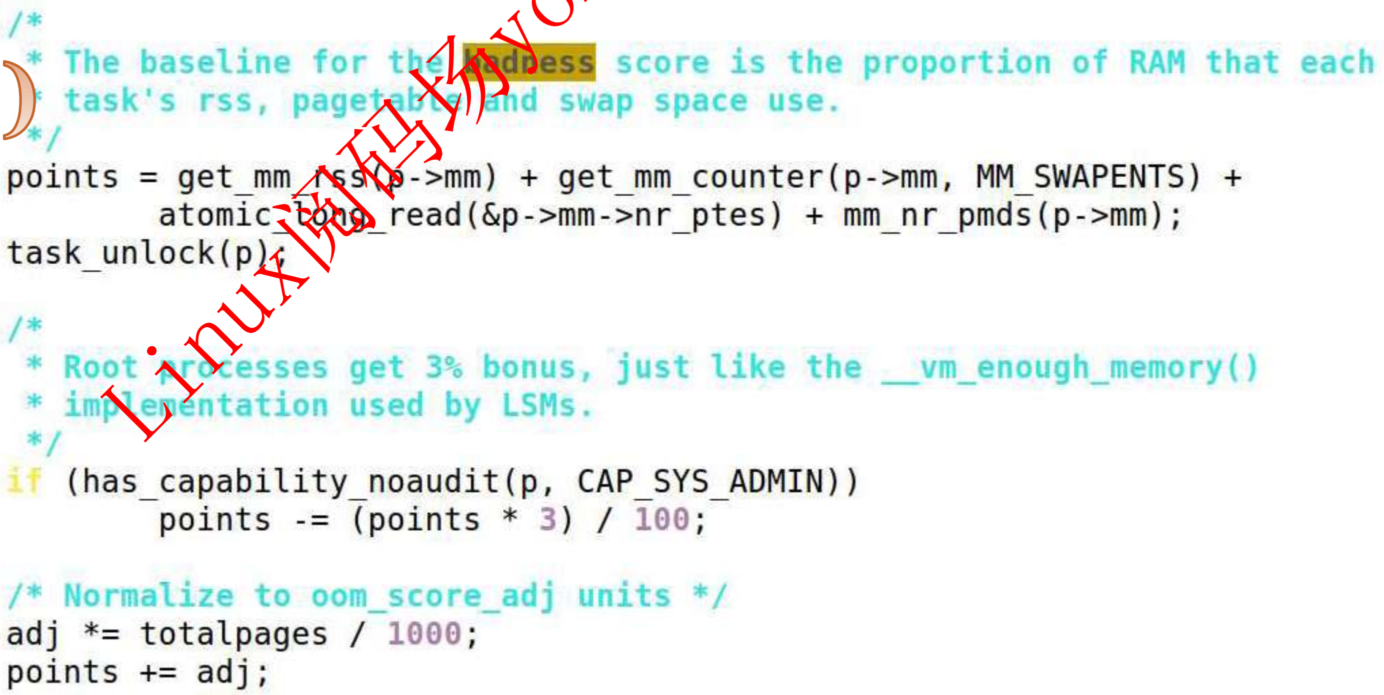
采用百分比乘以10（percent-times-ten）：一个使用全部内存的进程得分1000，使用0个字节的进程得分0。

✓ Root用户进程减去30

✓ oom_score_adj: oom_score会加上oom_score_adj这个值

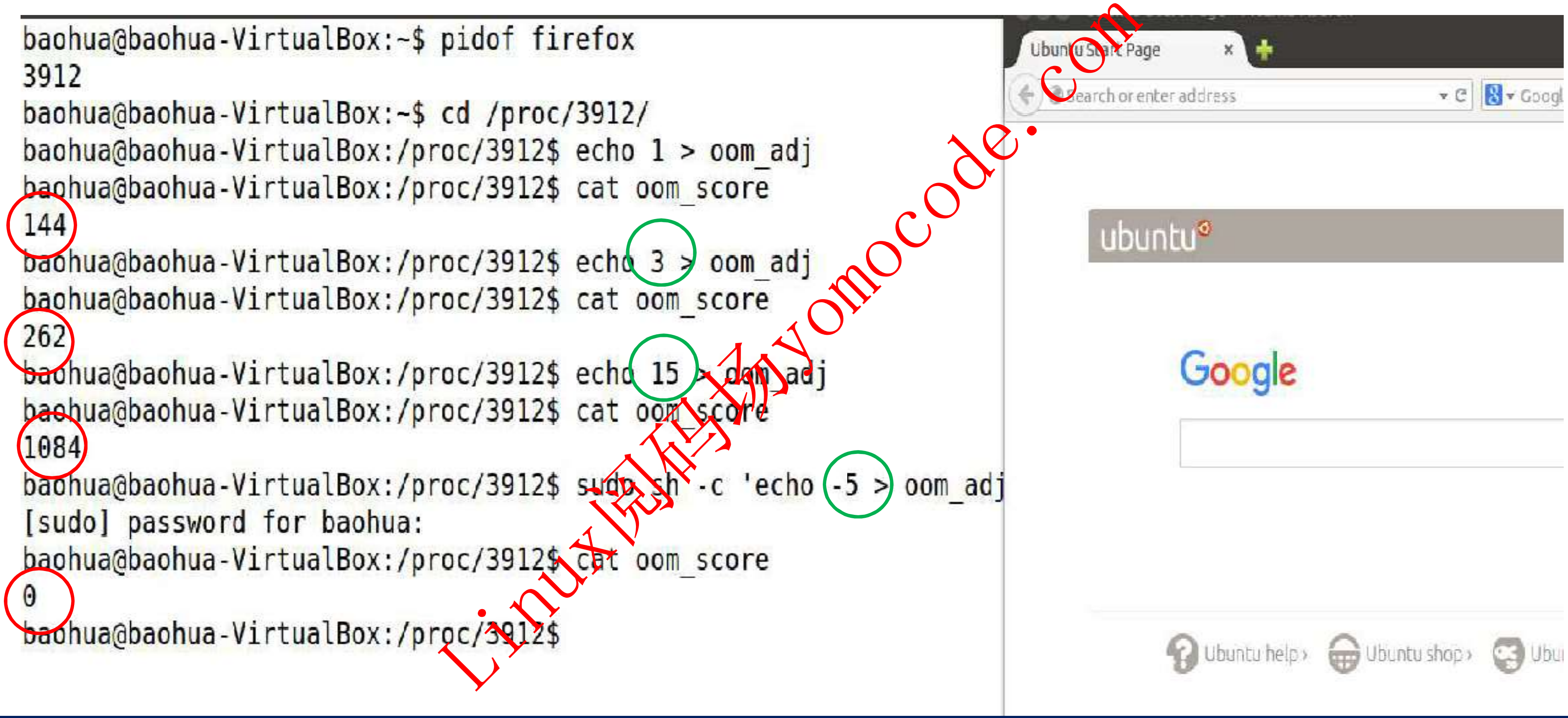
✓ oom_adj： -17~15的系数调整

badness()



```
/*
 * The baseline for the badness score is the proportion of RAM that each
 * task's rss, pagetable and swap space use.
 */
points = get_mm_rss(p->mm) + get_mm_counter(p->mm, MM_SWAPENTS) +
        atomic_long_read(&p->mm->nr_ptes) + mm_nr_pmds(p->mm);
task_unlock(p);

/*
 * Root processes get 3% bonus, just like the __vm_enough_memory()
 * implementation used by LSMs.
 */
if (has_capability_noaudit(p, CAP_SYS_ADMIN))
        points -= (points * 3) / 100;

/* Normalize to oom_score_adj units */
adj *= totalpages / 1000;
points += adj;
```

通过oom_adj调整进程的oom打分



```
baohua@baohua-VirtualBox:~$ pidof firefox
3912
baohua@baohua-VirtualBox:~$ cd /proc/3912/
baohua@baohua-VirtualBox:/proc/3912$ echo 1 > oom_adj
baohua@baohua-VirtualBox:/proc/3912$ cat oom_score
144
baohua@baohua-VirtualBox:/proc/3912$ echo 3 > oom_adj
baohua@baohua-VirtualBox:/proc/3912$ cat oom_score
262
baohua@baohua-VirtualBox:/proc/3912$ echo 15 > oom_adj
baohua@baohua-VirtualBox:/proc/3912$ cat oom_score
1084
baohua@baohua-VirtualBox:/proc/3912$ sudo sh -c 'echo -5 > oom_adj
[sudo] password for baohua:
baohua@baohua-VirtualBox:/proc/3912$ cat oom_score
0
baohua@baohua-VirtualBox:/proc/3912$
```

Android在程序退出时候，并不杀死进程，而是等OOM再杀死
Android根据不同的进程类型设置不同的oom_adj

课程练习源码

https://github.com/21cnbao/memory-courses

谢谢！