

第 0006 讲 1Linux 内核《内存布局和堆管理》

1、通过 cat /proc/cpuinfo 输出可访问地址空间，具体其它部分选项参数如下：

vendor_id: GenuineIntel // CPU 制造商

cpu family: 6 // CPU 产品代号

model: 142 // CPU 属于其系列当中的哪一个代号

model name: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz //

CPU 属于的名称、编号、主频

stepping: 12 // CPU 属于制作更新版本

cpu MHz: 2303.996 // CPUr 实际使用主频

cache size: 8192 KB // CPU 二级缓存

physical id: 0 // 单个 CPU 的标号

siblings: 1 // 单个 CPU 逻辑物理核数

core id: 0 // 当前物理核 在其所处的 CPU 中编号

cpu cores: 1 // 逻辑核所在 CPU 的物理核数

apicid: 0 // 用来区分不同逻辑核的编号

bogomips: 4607.99 // 系统内核启动时测算的 CPU 速度

clflush size: 64 // 每次刷新缓存的大小单位

cache_alignment: 64 // 缓存地址空间对齐单位

address sizes: 45 bits physical, 48 bits virtual // 可以访问的地址空间位数

2、通过 cat /proc/meminfo 输出系统架构内存分布情况， 具体如下：

```
root@ubuntu:/home/vico# cat /proc/meminfo
```

MemTotal: 12165668 kB // 所有可用内存空间的大小

MemFree: 10151616 kB // 系统还没有使用内存

MemAvailable: 10570548 kB // 真正系统可用内存

Buffers: 52596 kB // 专用用来给块设备做缓存的内存

Cached: 575008 kB // 分配给文件缓冲区的内存

SwapCached: 0 kB // 被高速缓冲缓存使用的交换空间
大小

Active: 1158808 kB // 使用高速缓冲存储器页面文件大
小

Inactive: 345196 kB // 没有经常使用的高速缓存存储器
大小

Active(anon): 870720 kB // 活跃的匿名内存

Inactive(anon): 12504 kB // 不活跃的匿名内存

Active(file): 288088 kB // 活跃的文件使用内存

Inactive(file): 332692 kB // 不活跃的文件使用内存

Unevictable: 16 kB // 不能被释放的内存页

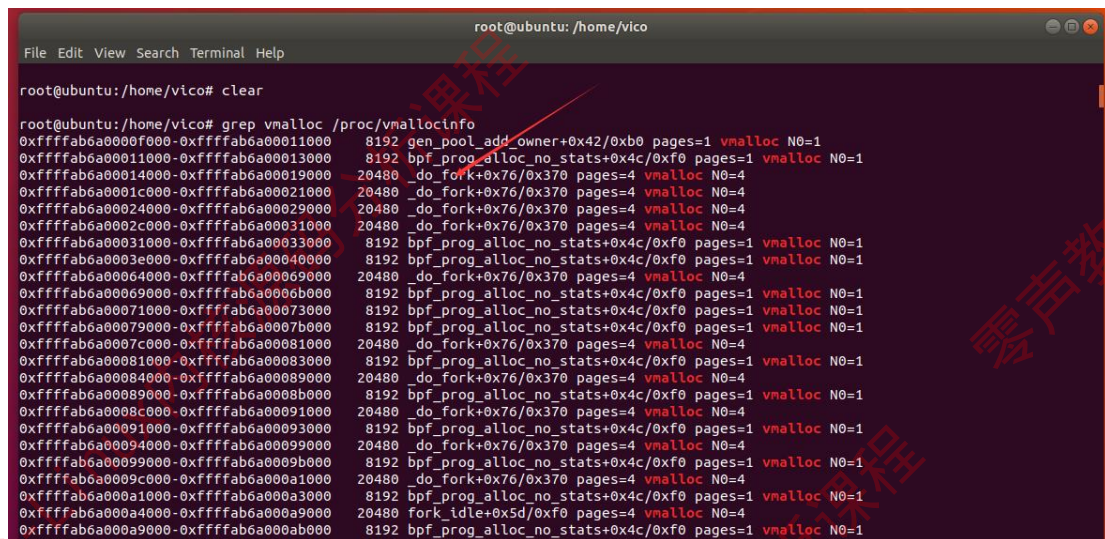
Mlocked: 16 kB // 系统调用 mlock 允许程序在物理内
存上锁住部分或全部地址空间

SwapTotal: 2097148 kB // 交换空间总内存大小
SwapFree: 2097148 kB // 交换空间空闲的内存大小
Dirty: 24 kB // 等待被写回到磁盘
Writeback: 0 kB // 正在被写加的大小
AnonPages: 876420 kB // 未映射页的内存/映射到用户空间的非文件页表大小
Mapped: 255420 kB // 映射文件内存
Shmem: 13896 kB // 已经被分配的共享内存
KReclaimable: 70884 kB // 可回收的 slab 内存
Slab: 151976 kB // 内存数据结构缓存大小
CommitLimit: 8179980 kB // 系统实际可以分配内存
Committed_AS: 4649264 kB // 系统当前已经分配的内存
VmallocTotal: 34359738367 kB // 预留虚拟内存的总量
VmallocUsed: 27836 kB // 已经被使用的虚拟内存
VmallocChunk: 0 kB // 可分配的最大逻辑地址连续的虚拟内存

3. Linux 内核动态内存分配通过系统接口实现

alloc_pages/__get_free_page: 以页为单位分配
vmalloc: 以字节为单位分配虚拟地址连续的内存块
kmalloc: 以字节为单位分配物理地址连续的内存块，它是
以 slab 为中心

我们也可以通过 `vmalloc` 分配的内存将它统计输出，具体如下：

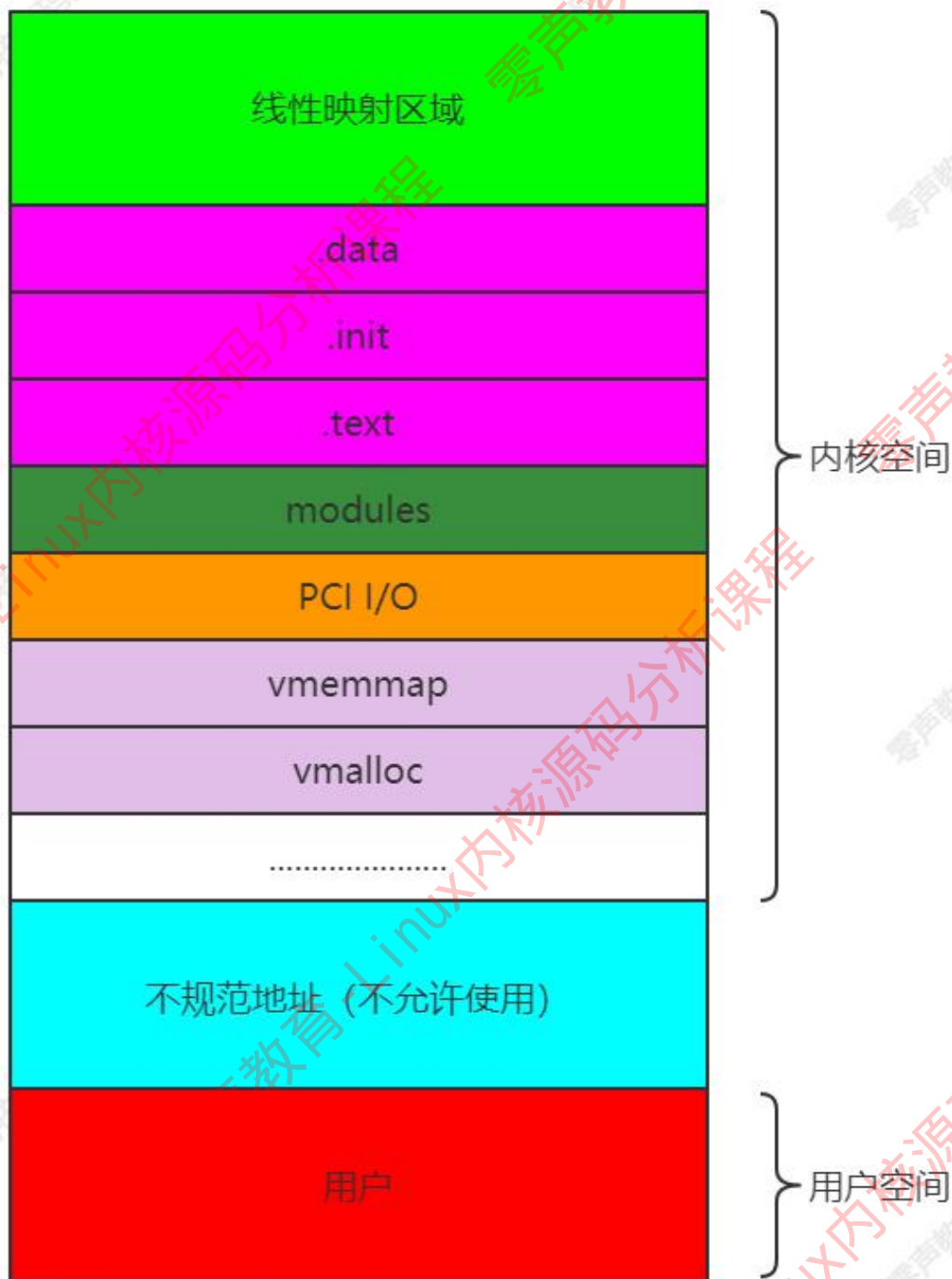


```
root@ubuntu: /home/vico
File Edit View Search Terminal Help

root@ubuntu: /home/vico# clear

root@ubuntu: /home/vico# grep vmalloc /proc/vmallocinfo
0xfffffab6a0000f000-0xfffffab6a00011000      8192 gen_pool_add owner+0x42/0xb0 pages=1 vmalloc N0=1
0xfffffab6a00011000-0xfffffab6a00013000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00014000-0xfffffab6a00019000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a0001c000-0xfffffab6a00021000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00024000-0xfffffab6a00029000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a0002c000-0xfffffab6a00031000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00031000-0xfffffab6a00033000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a0003e000-0xfffffab6a00040000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00064000-0xfffffab6a00069000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00069000-0xfffffab6a0006b000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00071000-0xfffffab6a00073000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00079000-0xfffffab6a0007b000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a0007c000-0xfffffab6a00081000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00081000-0xfffffab6a00083000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00084000-0xfffffab6a00089000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00089000-0xfffffab6a0008b000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a0008c000-0xfffffab6a00091000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00091000-0xfffffab6a00093000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a00094000-0xfffffab6a00099000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a00099000-0xfffffab6a0009b000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a0009c000-0xfffffab6a000a1000      20480 _do_fork+0x76/0x370 pages=4 vmalloc N0=4
0xfffffab6a000a1000-0xfffffab6a000a3000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
0xfffffab6a000a4000-0xfffffab6a000a9000      20480 fork_idle+0x5d/0xf0 pages=4 vmalloc N0=4
0xfffffab6a000a9000-0xfffffab6a000ab000      8192 bpf_prog_alloc_no_stats+0x4c/0xf0 pages=1 vmalloc N0=1
```

4、Linux 内核内存布局（ARM64 架构处理器内存分布图），具体如下：



ARM64架构处理器Linux内核内存分布图

KASAN (影子区)：它是一个动态检测内存错误的工具，原理利用额外的内存标记可用内存的状态（将 $1/8$ 内存作为影子区）

modules：内核模块使用的虚拟地址空间

vmalloc: vmalloc 函数使用的虚拟地址空间

.text: 代码段

.init: 模块初始化数据

.data: 数据段

.bss: 静态内存分配段

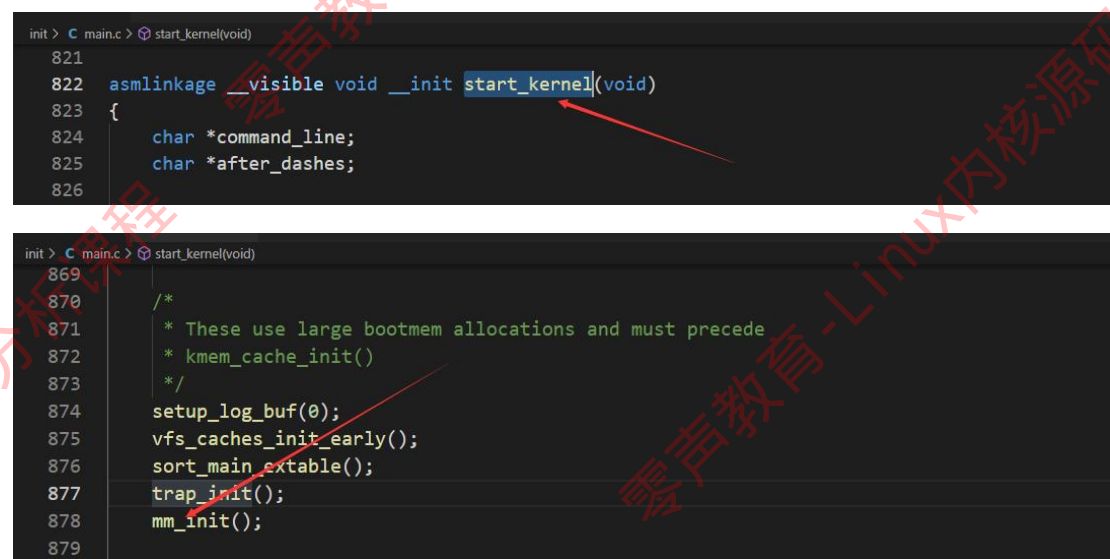
fixed: 固定映射区域

PCI I/O: PCI 设备的 I/O 地址空间

vmemmap: 内存的物理地址如果不连续的话，就会存在内存空洞（稀疏内存）vmemmap 就用来存放稀疏内存的 page 结构体的数据的虚拟地址空间

memory: 线性映射区域

5、我们可以通过内存布局打印输出（Linux 内核初始化完成后，整体布局稳定。通过 Vexpress 平台输出即可）：



```
init > C main.c > start_kernel(void)
821
822  asmlinkage __visible void __init start_kernel(void)
823  {
824      char *command_line;
825      char *after_dashes;
826
869
870      /*
871       * These use large bootmem allocations and must precede
872       * kmem_cache_init()
873       */
874      setup_log_buf(0);
875      vfs_caches_init_early();
876      sort_main_extable();
877      trap_init();
878      mm_init();
879
```



```

init > C main.c > mm_init(void)
792 /*
793  * Set up kernel memory allocators
794  */
795 static void __init mm_init(void)
796 {
797     /*
798      * page_ext requires contiguous pages,
799      * bigger than MAX_ORDER unless SPARSEMEM.
800      */
801     page_ext_init_flatmem();
802     init_debug_pagealloc();
803     report_meminit();
804     mem_init();
805     kmem_cache_init();
806     kmemleak_init();
807     pgtable_init();
808     debug_objects_mem_init();
809     vmalloc_init();
810     ioremap_huge_init();
811     /* Should be run before the first non-init thread is created */
812     init_espfix_bsp();
813     /* Should be run after espfix64 is set up. */
814     pti_init();
815 }

```

```

arch > x86 > mm > C init_32.c > mem_init(void)
766 void __init mem_init(void)
767 {
768     pci_iommu_alloc();
769
770 #ifdef CONFIG_FLATMEM
771     BUG_ON(!mem_map);
772 #endif

```

```

arch > x86 > mm > C init_32.c > mem_init(void)
787 after_bootmem = 1;
788 x86_init.hyper.init_after_bootmem();
789
790 mem_init_print_info(NULL);
791 printk(KERN_INFO "virtual kernel memory layout:\n"
792         "  fixmap : 0x%08lx - 0x%08lx (%4ld kB)\n"
793         "  cpu_entry : 0x%08lx - 0x%08lx (%4ld kB)\n"
794 #ifdef CONFIG_HIGHMEM
795         "  pkmap : 0x%08lx - 0x%08lx (%4ld kB)\n"
796 #endif
797         "  vmalloc : 0x%08lx - 0x%08lx (%4ld MB)\n"
798         "  lowmem : 0x%08lx - 0x%08lx (%4ld MB)\n"
799         "  .init : 0x%08lx - 0x%08lx (%4ld kB)\n"
800         "  .data : 0x%08lx - 0x%08lx (%4ld kB)\n"
801         "  .text : 0x%08lx - 0x%08lx (%4ld kB)\n",
802         FIXADDR_START, FIXADDR_TOP,
803         (FIXADDR_TOP - FIXADDR_START) >> 10,
804
805         CPU_ENTRY_AREA_BASE,
806         CPU_ENTRY_AREA_BASE + CPU_ENTRY_AREA_MAP_SIZE,
807         CPU_ENTRY_AREA_MAP_SIZE >> 10,
808
809 #ifdef CONFIG_HIGHMEM
810         PKMAP_BASE, PKMAP_BASE+LAST_PKMAP*PAGE_SIZE,
811         (LAST_PKMAP*PAGE_SIZE) >> 10,
812 #endif

```

6、内存描述 mm_struct 结构体当中，有堆起始地址和结束的成员，具体内核源码如下：

```
include > linux > C mm_types.h > mm_struct
376
377
378 struct mm_struct {
379     struct {
380         struct vm_area_struct *mmap;          /* list of VMAs */
381         struct rb_root mm_rb;
382         u64 vmacache_seqnum;                  /* per-thread vmacache */
383 #ifdef CONFIG_MMU
384         unsigned long (*get_unmapped_area)(struct file *filp,
385         unsigned long addr, unsigned long len,
386         unsigned long pgoff, unsigned long flags);
387 #endif
388
389     unsigned long total_vm;          /* Total pages mapped */
390     unsigned long locked_vm;         /* Pages that have PG_mlocked set */
391     atomic64_t pinned_vm;            /* Refcount permanently increased */
392     unsigned long data_vm;           /* VM_WRITE & ~VM_SHARED & ~VM_STACK */
393     unsigned long exec_vm;           /* VM_EXEC & ~VM_WRITE & ~VM_STACK */
394     unsigned long stack_vm;          /* VM_STACK */
395     unsigned long def_flags;
396
397     spinlock_t arg_lock; /* protect the below fields */
398     unsigned long start_code, end_code, start_data, end_data;
399     unsigned long start_brk, brk, start_stack;
400     unsigned long arg_start, arg_end, env_start, env_end;
401
402     /* ... other fields ... */
403 };
```

brk 系统调用内核源码如下：

```
mm > C mmap.c > SYSCALL_DEFINE1(brk, unsigned long, brk)
192 SYSCALL_DEFINE1(brk, unsigned long, brk)
193 {
194     unsigned long retval;
195     unsigned long newbrk, oldbrk, origbrk;
196     struct mm_struct *mm = current->mm;
197     struct vm_area_struct *next;
198     unsigned long min_brk;
199     bool populate;
200     bool downgraded = false;
201     LIST_HEAD(uf);
202
203     if (down_write_killable(&mm->mmap_sem))
204         return -EINTR;
205
206     origbrk = mm->brk;
207     newbrk = brk;
```

Linux 系统当中有两个方法可以创建堆：

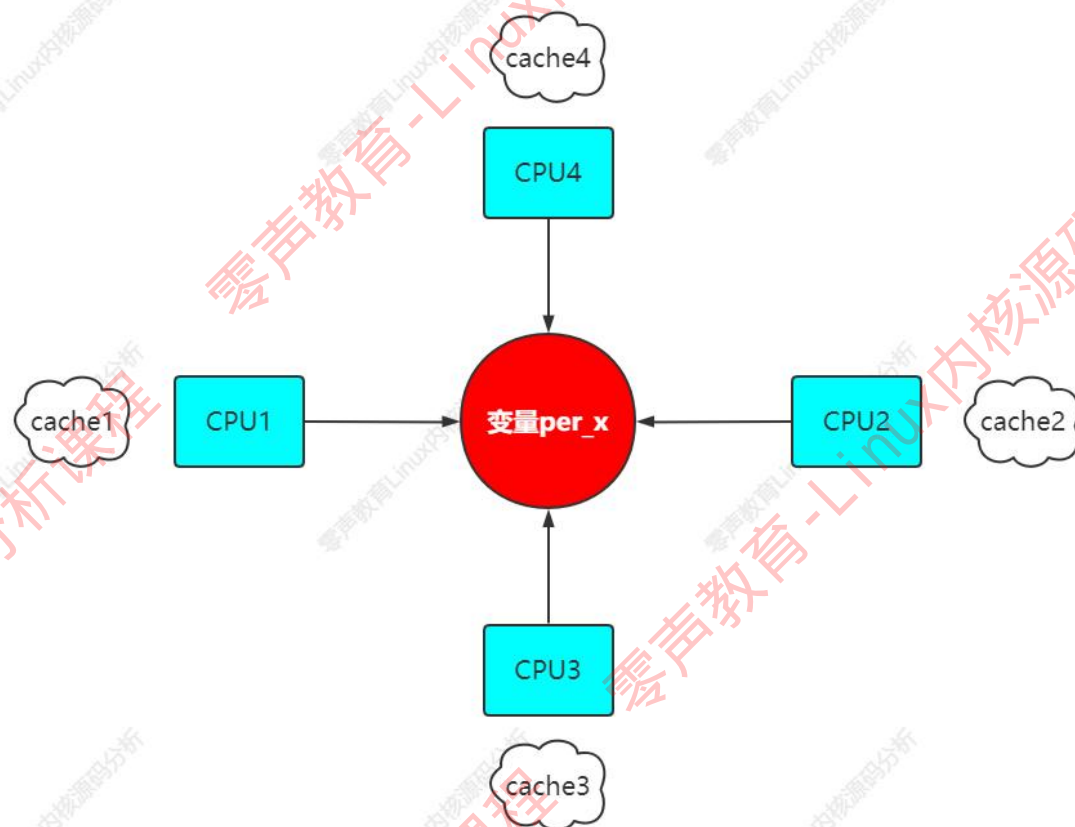
brk()是系统调用，实际是设置进程数据段的结束地址，将数据段的结束地址向高地址移动。

mmap()向操作系统申请一段虚拟地址空间（使用映射到某个文件）。当不用此空间来映射到某个文件时，这块空间称为匿名空间可以用来作为堆空间。

7、per-CPU 计数器

```
include > linux > C percpu_counter.h > 68 percpu_counter
18 #ifndef CONFIG_SMP
19
20 struct percpu_counter {
21     raw_spinlock_t lock;
22     s64 count;
23 #ifdef CONFIG_HOTPLUG_CPU
24     struct list_head list; /* All percpu_counters are on a list */
25 #endif
26     s32 __percpu *counters;
27 };
28
```

案例分析如下：



核源码分析课程

零声教育-Linux

零声教育-Linux内核