# 第 0019 讲 2 伙伴系统算法案例分析

# 内存管理专题--2 伙伴系统算法案例分析

## 一、伙伴系统算法原理

Linux 的伙伴算法把所有的空闲页面分为 11 个块组，每组中块的大小是 2 的幂次方个页面。例如，第 0 组中块的大小都为 $2^0$（1 个页面），第 1 组中块的大小都为 $2^1$（2 个页面），第 8 组中块的大小都为 $2^8$（256 个页面），依次类推。也就是说，每一组中块的大小是相同的，且这同样大小的块形成一个链表。

系统内存中的每个物理内存页（页帧），都对应于一个 struct page 实例，每个内存域都关联了一个 struct zone 的实例，其中保存了用于管理伙伴数据的主要数组，具体如下：

include > linux > C mmzone.h > zone_type

```
361
362 struct zone {
363     /* Read-mostly fields */
364
365     /* zone watermarks, access with *_wmark_pages(zone) macros */
366     unsigned long _watermark[NR_WMARK];
367     unsigned long watermark_boost;
368
369     unsigned long nr_reserved_highatomic;
370
```

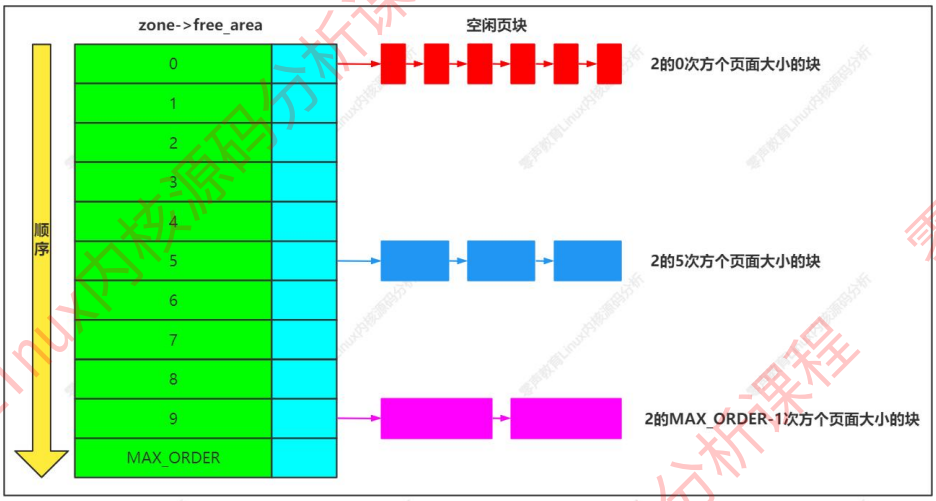
include > linux > C mmzone.h > zone_type

```
456     /* Write-intensive fields used from the page allocator */
457     ZONE_PADDING(_pad1_)
458
459     /* free areas of different sizes */
460     struct free_area    free_area[MAX_ORDER];
461
462     /* zone flags, see below */
463     unsigned long       flags;
464
```

include > linux > C mmzone.h > ...

```
29
30 /* Free memory management - zoned buddy allocator.  */
31 #ifndef CONFIG_FORCE_MAX_ZONEORDER
32 #define MAX_ORDER 11
33 #else
34 #define MAX_ORDER CONFIG_FORCE_MAX_ZONEORDER
35 #endif
36 #define MAX_ORDER_NR_PAGES (1 << (MAX_ORDER - 1))
37
```

一般来说 MAX_ORDER 默认定义为 11，但如果特定于体系结构的代码设置 FORCE_MAX_ZONEORDER 配置选项，该值也可以手工改变。

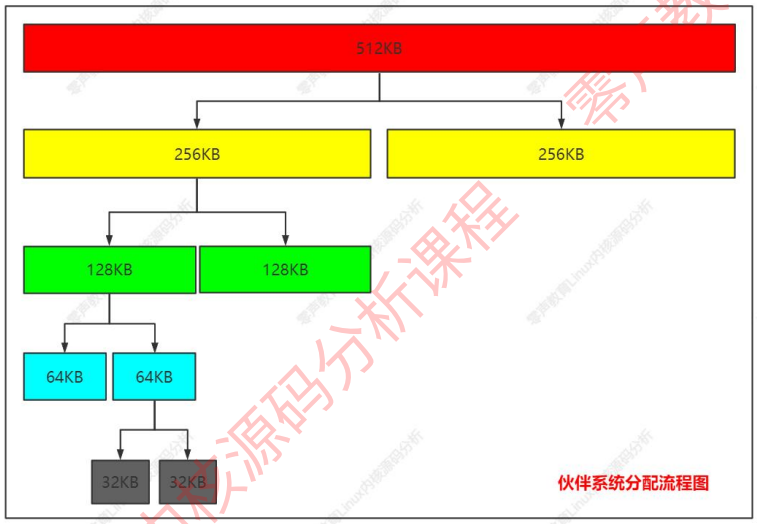



struct free_area 是一个伙伴系统的辅助数据结构：

```
include > linux > C mmzone.h > free_area
 96  struct free_area {
 97      struct list_head    free_list[MIGRATE_TYPES];
 98      unsigned long       nr_free;
 99  };
100
```

其中：free_list 是用于连接空闲页的链表。页链表包含大小相同的连续内存区；nr_free 指定了当前内存区中空闲页块的数目（对 0 阶内存区逐页计算，对 1 阶内存区计算页对的数目，对 2 阶内存区计算 4 页集合的数目，依次类推。伙伴系统的分配器维护空闲页面所组成的块，这里每一块都是 2 的方幂个页面，方幂的指数称为阶。

伙伴系统从物理连续的大小固定的段进行分配，方法如上所述，内核请求 28KB 的内存，具体案例分析如下：




## 二、分配页和释放页

1、伙伴系统设计思路和申请和释放 API 基本方式一样，Linux 内核中常用的分配物理内存页面的接口是 alloc_pages()，分配一个或多个连续的物理页面，分配原则个数为 2^n。相对于多次离散物理页面它可以提高系统内存的碎片化。具体源码分析如下：

include > linux > C gfp.h > alloc_pages(gfp_t, unsigned int)

```
489  /*
490   * Allocate pages, preferring the node given as nid. When nid == NUMA_NO_NODE,
491   * prefer the current CPU's closest node. Otherwise node must be valid and
492   * online.
493   */
494  static inline struct page *alloc_pages_node(int nid, gfp_t gfp_mask,
495                                              unsigned int order)
496  {
497      if (nid == NUMA_NO_NODE)
498          nid = numa_mem_id();
499
500      return __alloc_pages_node(nid, gfp_mask, order);
501  }
502
```

如上面函数只是简要检查防止申请内存过大，如果指定节点不存在，Linux 内核自动使用当前执行 CPU 对应的节点 ID，最后调度核心函数__alloc_pages_nodemask(...)。

include > linux > C gfp.h > __alloc_pages_node(int, gfp_t, unsigned int)

```
476  /*
477   * Allocate pages, preferring the node given as nid. The node must be valid and
478   * online. For more general interface, see alloc_pages_node().
479   */
480  static inline struct page *
481  __alloc_pages_node(int nid, gfp_t gfp_mask, unsigned int order)
482  {
483      VM_BUG_ON(nid < 0 || nid >= MAX_NUMNODES);
484      VM_WARN_ON((gfp_mask & __GFP_THISNODE) && !node_online(nid));
485
486      return __alloc_pages(gfp_mask, order, nid);
487  }
488
```

include > linux > C gfp.h > ...

```
470  static inline struct page *
471  __alloc_pages(gfp_t gfp_mask, unsigned int order, int preferred_nid)
472  {
473      return __alloc_pages_nodemask(gfp_mask, order, preferred_nid, NULL);
474  }
```

```
4504  struct page *
4505  __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order, int preferred_nid,
4506                         nodemask_t *nodemask)
4507  {
4508      struct page *page;
4509      unsigned int alloc_flags = ALLOC_WMARK_LOW;
4510      gfp_t alloc_mask; /* The gfp_t that was actually used for allocation */
4511      struct alloc_context ac = { };
4512
4513      /*
4514       * There are several places where we assume that the order value is sane
4515       * so bail out early if the request is out of bound.
4516       */
4517      if (unlikely(order >= MAX_ORDER)) {
4518          WARN_ON_ONCE(!(gfp_mask & __GFP_NOWARN));
4519          return NULL;
4520      }
```

2、伙伴系统释放页框步骤（free_pages()）：

mm > C page_alloc.c > EXPORT_SYMBOL(__free_pages)

```
4603
4604  void __free_pages(struct page *page, unsigned int order)
4605  {
4606      if (put_page_testzero(page))
4607          free_the_page(page, order);
4608  }
4609  EXPORT_SYMBOL(__free_pages);
4610
4611
```

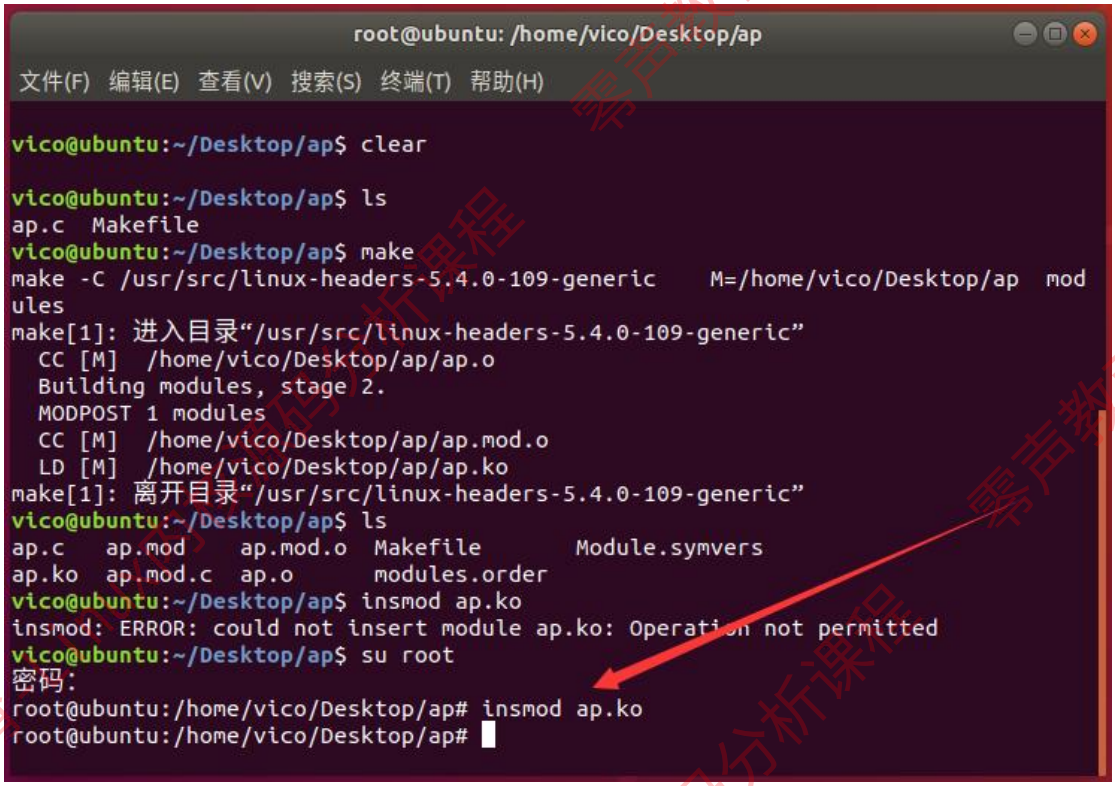
mm > C page_alloc.c

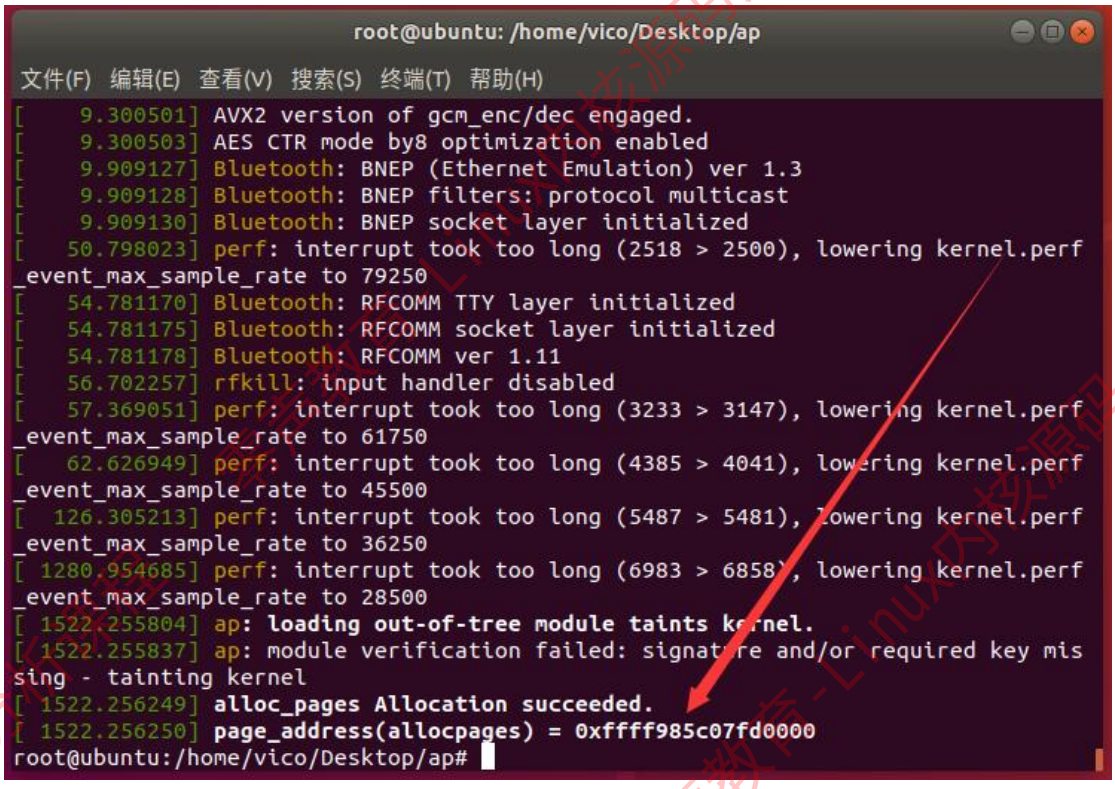
```
4595  static inline void free_the_page(struct page *page, unsigned int order)
4596  {
4597      if (order == 0)      /* Via pcp? */
4598          free_unref_page(page);
4599      else
4600          __free_pages_ok(page, order);
4601  }
4602
4603
```

mm > C page_alloc.c

```
1292
1293  static void __free_pages_ok(struct page *page, unsigned int order)
1294  {
1295      unsigned long flags;
1296      int migratetype;
1297      unsigned long pfn = page_to_pfn(page);
1298
1299      if (!free_pages_prepare(page, order, true))
1300          return;
1301
1302      migratetype = get_pfnblock_migratetype(page, pfn);
1303      local_irq_save(flags);
1304      __count_vm_events(PGFREE, 1 << order);
1305      free_one_page(page_zone(page), page, pfn, order, migratetype);
1306      local_irq_restore(flags);
1307  }
1308
```

## 三、伙伴系统实现


root@ubuntu: /home/vico/Desktop/ap
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
vico@ubuntu:~/Desktop/ap$ clear
vico@ubuntu:~/Desktop/ap$ ls
ap.c Makefile
vico@ubuntu:~/Desktop/ap$ make
make -C /usr/src/linux-headers-5.4.0-109-generic M=/home/vico/Desktop/ap modules
make[1]: 进入目录“/usr/src/linux-headers-5.4.0-109-generic”
CC [M] /home/vico/Desktop/ap/ap.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/vico/Desktop/ap/ap.mod.o
LD [M] /home/vico/Desktop/ap/ap.ko
make[1]: 离开目录“/usr/src/linux-headers-5.4.0-109-generic”
vico@ubuntu:~/Desktop/ap$ ls
ap.c ap.mod ap.mod.o Makefile Module.symvers
ap.ko ap.mod.c ap.o modules.order
vico@ubuntu:~/Desktop/ap$ insmod ap.ko
insmod: ERROR: could not insert module ap.ko: Operation not permitted
vico@ubuntu:~/Desktop/ap$ su root
密码:
root@ubuntu:/home/vico/Desktop/ap# insmod ap.ko
root@ubuntu:/home/vico/Desktop/ap#


root@ubuntu: /home/vico/Desktop/ap
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[ 9.300501] AVX2 version of gcm_enc/dec engaged.
[ 9.300503] AES CTR mode by8 optimization enabled
[ 9.909127] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 9.909128] Bluetooth: BNEP filters: protocol multicast
[ 9.909130] Bluetooth: BNEP socket layer initialized
[ 50.798023] perf: interrupt took too long (2518 > 2500), lowering kernel.perf_event_max_sample_rate to 79250
[ 54.781170] Bluetooth: RFCOMM TTY layer initialized
[ 54.781175] Bluetooth: RFCOMM socket layer initialized
[ 54.781178] Bluetooth: RFCOMM ver 1.11
[ 56.702257] rfkill: input handler disabled
[ 57.369051] perf: interrupt took too long (3233 > 3147), lowering kernel.perf_event_max_sample_rate to 61750
[ 62.626949] perf: interrupt took too long (4385 > 4041), lowering kernel.perf_event_max_sample_rate to 45500
[ 126.305213] perf: interrupt took too long (5487 > 5481), lowering kernel.perf_event_max_sample_rate to 36250
[ 1280.954685] perf: interrupt took too long (6983 > 6858), lowering kernel.perf_event_max_sample_rate to 28500
[ 1522.255804] ap: loading out-of-tree module taints kernel.
[ 1522.255837] ap: module verification failed: signature and/or required key missing - tainting kernel
[ 1522.256249] alloc_pages Allocation succeeded.
[ 1522.256250] page_address(allocpages) = 0xffff985c07fd0000
root@ubuntu:/home/vico/Desktop/ap#