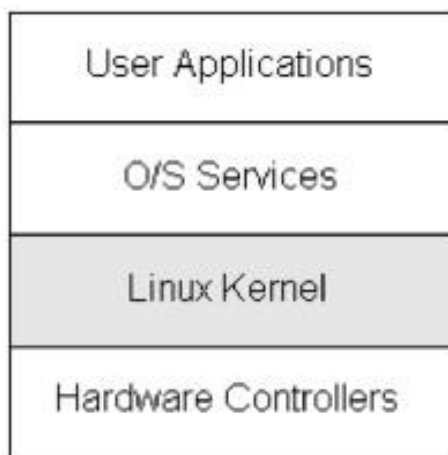




第011讲 伙伴分配器



零声学院讲师: Vico老师



一、分区的伙伴分配器

二、分配页

三、释放页



当系统内核初始化完毕后，使用页分配器管理物理页，当使用的页分配器是伙伴分配器，伙伴分配器的特点是算法简单且高效。

连续的物理页称为页块（page block）。阶（order）是伙伴分配器的一个专业术语，是页的数量单位， 2^n 个连续页称为n阶页块。

满足以下条件的两个n阶页块称为伙伴（buddy --> 英 ['bʌdi]）：

- 1、两个页块是相邻的，即物理地址是连续的；
- 2、页块的第一页的物理页号必须是 2^n 的整数倍；
- 3、如果合并成 $(n+1)$ 阶页块，第一页的物理页号必须是 2^{n+1} 的整数倍。



伙伴分配器分配和释放物理页的数量单位为阶。分配 n 阶页块的过程如下：

- 1、查看是否有空闲的 n 阶页块，如果有直接分配；否则，继续执行下一步；
- 2、查看是否存在空闲的 $(n+1)$ 阶页块，如果有，把 $(n+1)$ 阶页块分裂为两个 n 阶页块，一个插入空闲 n 阶页块链表，另一个分配出去；否则继续执行下一步。
- 3、查看是否存在空闲的 $(n+2)$ 阶页块，如果有把 $(n+2)$ 阶页块分裂为两个 $(n+1)$ 阶页块，一个插入空闲 $(n+1)$ 阶页块链表，另一个分裂为两个 n 阶页块，一个插入空闲 n 阶页块链表，另一个分配出去；如果没有，继续查看更高阶是否存在空闲页块。



一、分区的伙伴分配器

1、数据结构

分区的伙伴分配器专注于某个内存节点的某个区域。内存区域的结构体成员free_area用来维护空闲页块，数组下标对应页块的阶数。 内核源码结构：

```
C mmzone.h X
include > linux > C mmzone.h > zone
308
369 #ifndef __GENERATING_BOUNDS_H
370
371 struct zone {
372     /* Read-mostly fields */
373
473     /* free areas of different sizes */
474     struct free_area    free_area[MAX_ORDER]; // 不同长度的空闲区域
448     unsigned long      managed_pages; // 伙伴分配器管理的物理页的数量
23 #ifndef CONFIG_FORCE_MAX_ZONEORDER
24 #define MAX_ORDER 11
25 #else
26 #define MAX_ORDER CONFIG_FORCE_MAX_ZONEORDER
27 #endif
```



free_area结构体内核源码如下:

```
95 struct free_area {
96     struct list_head    free_list[MIGRATE_TYPES];
97     unsigned long        nr_free;
98 };

--
62 #end (enum migratetype)MIGRATE_TYPES = 4
63 MIGRATE_TYPES
```



2、根据分配标志获取首选区域类型

申请页时，最低的4个标志位用来指定首选的内存区域类型，内核源码如下：

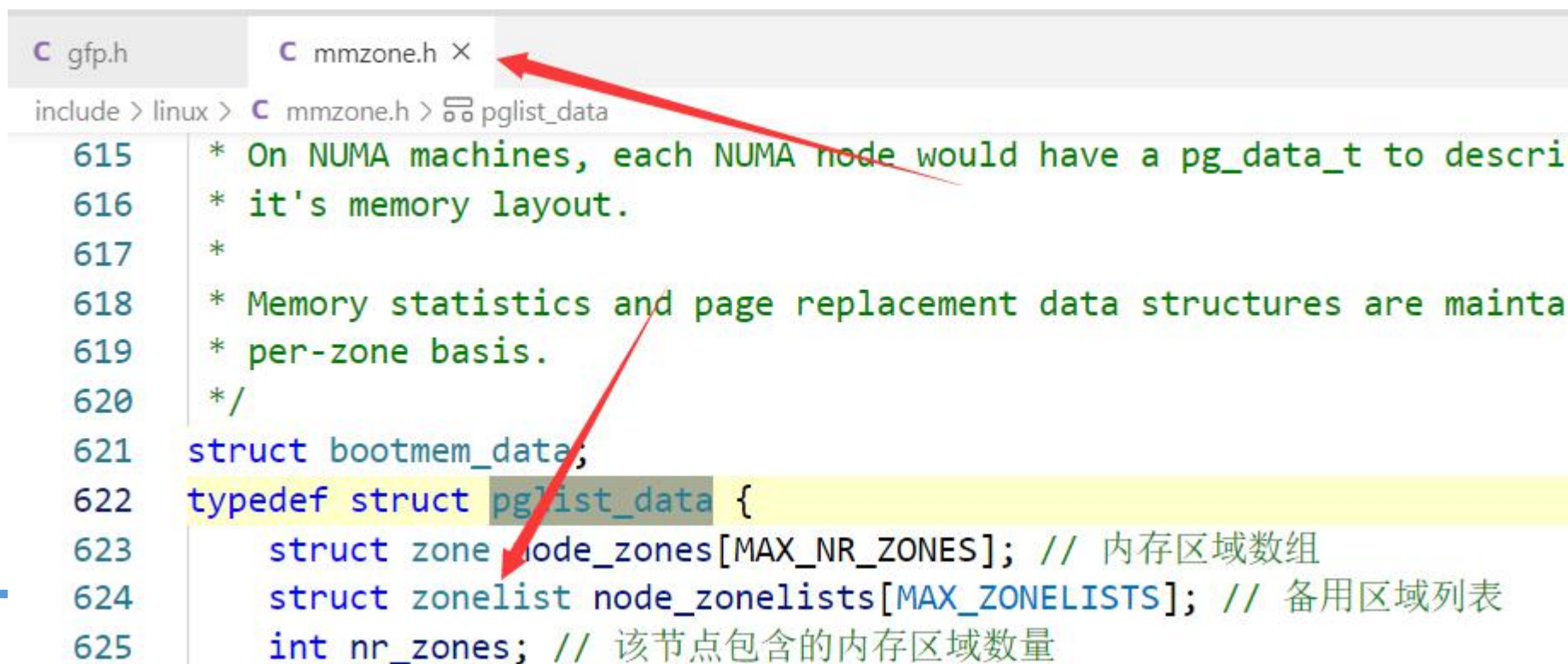
```
C gfp.h ×
include > linux > C gfp.h > ...
1  #ifndef __LINUX_GFP_H
2  #define __LINUX_GFP_H
3
18 #define __GFP_DMA          0x01u
19 #define __GFP_HIGHMEM      0x02u
20 #define __GFP_DMA32        0x04u
21 #define __GFP_MOVABLE      0x08u
```




3. 备用区域列表

如果首选的内存节点或区域不能满足分配请求，可以从备用的内存区域借用物理页。
借用必须遵守相应的规则。

内存节点的pg_data_t实例已定义备用区域列表，内核源码如下：



```
include > linux > C mmzone.h > pglist_data
615  * On NUMA machines, each NUMA node would have a pg_data_t to descri
616  * it's memory layout.
617  *
618  * Memory statistics and page replacement data structures are mainta
619  * per-zone basis.
620  */
621 struct bootmem_data;
622 typedef struct pglist_data {
623     struct zone node_zones[MAX_NR_ZONES]; // 内存区域数组
624     struct zonelist node_zonelists[MAX_ZONELISTS]; // 备用区域列表
625     int nr_zones; // 该节点包含的内存区域数量
```




```
566 enum {
567     ZONELIST_FALLBACK, /* zonelist with fallback */
568 #ifdef CONFIG_NUMA
569     /*
570      * The NUMA zonelists are doubled because we need zonelists that
571      * restrict the allocations to a single node for __GFP_THISNODE.
572      */
573     ZONELIST_NOFALLBACK, /* zonelist without fallback (__GFP_THISNODE) */
574 #endif
575     MAX_ZONELISTS
576 };

582 struct zoneref {
583     struct zone *zone; /* Pointer to actual zone */
584     int zone_idx; /* zone_idx(zoneref->zone) */
585 };
```



4、区域水线

首选的内存区域什么情况下从备用区域借用物理页呢？每个内存区域有3个水线

- a.高水线 (high)：如果内存区域的空闲页数大于高水线，说明内存区域的内存充足；
- b.低水线 (low)：如果内存区域的空闲页数小于低水线，说明内存区域的内存轻微不足；
- c.最低水线 (min)：如果内存区域的空闲页数小于最低水线，说明内存区域的内存严重不足。

```

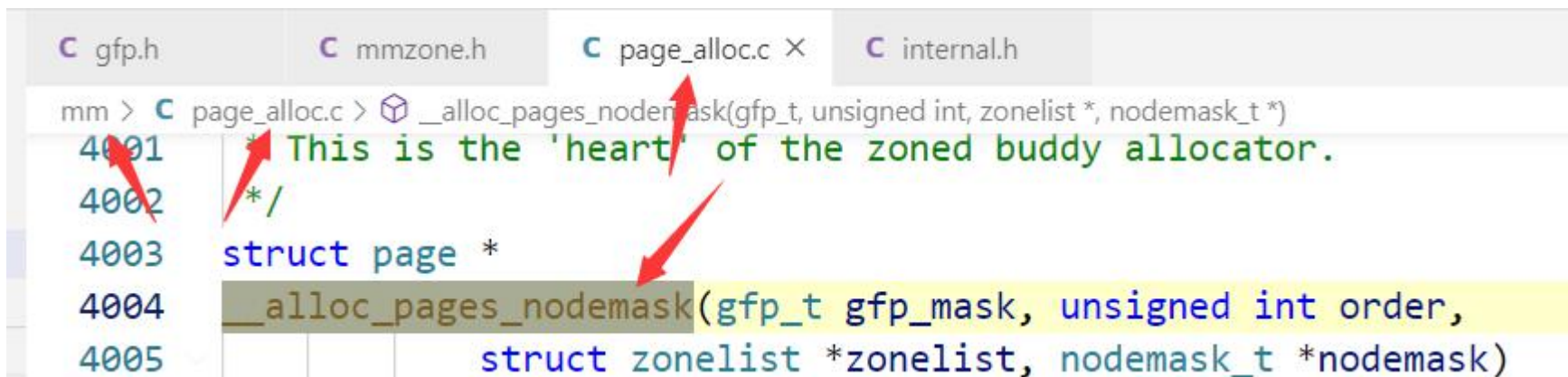
C gfp.h      C mmzone.h X
include > linux > C mmzone.h > zone_watermarks
254
255 enum zone_watermarks {
256     WMARK_MIN,          371 struct zone {
257     WMARK_LOW,          372     /* Read-mostly fields */
258     WMARK_HIGH,         373
259     NR_WMARK             374     /* zone watermarks, access with *_wmark_pages(zone) macros */
260 };                        375     unsigned long watermark[NR_WMARK]; // 页分配器使用的水线
261                        376


```



二、分配页

在Linux内核中，所有分配页的函数最终都会调用到__alloc_pages_nodemask，此函数被称为**分区的伙伴分配器**的心脏。函数原型如下：



```
mm > C page_alloc.c >  __alloc_pages_nodemask(gfp_t, unsigned int, zonelist *, nodemask_t *)
4001  * This is the 'heart' of the zoned buddy allocator.
4002  */
4003  struct page *
4004  __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
4005                        struct zonelist *zonelist, nodemask_t *nodemask)
```

算法流程：

- 1、根据分配标志位得到首选区域类型和迁移类型；
- 2、执行**快速路径**，使用低水线尝试第一次分配；
- 3、如果快速路径分配失败，才执行**慢速路径**。



页分配器内部分配标志位:

```
mm > C internal.h
474  /* The ALLOC_WMARK bits are used as an index to zone->watermark */
475  #define ALLOC_WMARK_MIN      WMARK_MIN
476  #define ALLOC_WMARK_LOW      WMARK_LOW
477  #define ALLOC_WMARK_HIGH      WMARK_HIGH
478  #define ALLOC_NO_WATERMARKS 0x04 /* don't check watermarks at all */
```



1、快速路径调用函数如下:

```
C gfp.h  C mmzone.h  C page_alloc.c X  C internal.h
mm > C page_alloc.c > get_page_from_freelist(gfp_t, unsigned int, int, const alloc_context *)
3015  ^ a page.
3016  */
3017  static struct page *
3018  get_page_from_freelist(gfp_t gfp_mask, unsigned int order, int alloc_flags,
3019                        const struct alloc_context *ac)
3020  {
3021      struct zoneref *z = ac->preferred_zoneref;
3022      struct zone *zone;
3023      struct pglist_data *last_pgdat_dirty_limit = NULL;
3024  }
```




2、慢速路径调用函数如下:

```
C gfp.h  C mmzone.h  C page_alloc.c X  C internal.h
mm > C page_alloc.c > __alloc_pages_slowpath(gfp_t, unsigned int, alloc_context *)
3676 static inline struct page *
3677 __alloc_pages_slowpath(gfp_t gfp_mask, unsigned int order,
3678                        struct alloc_context *ac)
3679 {
3680     bool can_direct_reclaim = gfp_mask & __GFP_DIRECT_RECLAIM;
3681     const bool costly_order = order > PAGE_ALLOC_COSTLY_ORDER;
```



三、释放页



页分配器提供释放页的接口:

`void __free_pages(struct page *page, unsigned int order)`, 第一个参数是第一个物理页的page实例的地址, 第二个参数是阶数。

```
mm > C page_alloc.c >  __free_pages(page *, unsigned int)
4082
4083 void __free_pages(struct page *page, unsigned int order)
4084 {
4085     if (put_page_testzero(page)) {
4086         if (order == 0)
4087             free_hot_cold_page(page, false);
4088         else
4089             __free_pages_ok(page, order);
4090     }
4091 }
4092
```




办学宗旨：一切只为渴望更优秀的你

办学愿景：让技术简单易懂