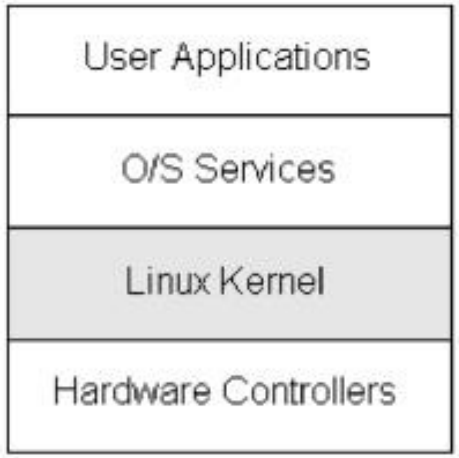
第0016讲 1页回收机制（一）



零声学院讲师：Vico老师

一、数据结构

二、发起页回收

当我们申请分配页的时候，页分配器首先尝试使用低水线分配页。如果使用低水线分配失败，说明内存轻微不足，页分配器将会唤醒内存节点的页回收内核线程，异步回收页，然后尝试使用最低水线分配页。如果使用最低水线分配失败，说明内存严重不足，页分配器会直接回收。

针对不同的物理页，采用不同的回收策略：交换支持的页和存储设备支持的文件页。

那我们根据什么的原则选择回收物理页呢?

Linux内核使用LRU（Least Recently Used，最近最少使用）算法选择最近最少使用的物理页。

回收物理页的时候，如果物理页被映射到进程的虚拟地址空间，那么需要从页表中删除虚拟页到物理页的映射。如何知道物理页被映射到哪些虚拟页?

1、LRU（最近最少使用）链表

页回收算法使用LRU算法选择回收的页。每个内存节点的pglist_data实例就有一个成员lruvec，称为LRU向量，LRU向量包含5条LRU链表。

include > linux > C mmzone.h > pglist_data

```
601  typedef struct pglist_data { // 内存节点的实例
602      struct zone node_zones[MAX_NR_ZONES];
603      struct zonelist node_zonelists[MAX_ZONELISTS];
604      int nr_zones;

690      /* Fields commonly accessed by the page reclaim scanner */
691      struct lruvec        lruvec;
692
```

struct lruvec结构定义如下：

mmzone.h 4

include > linux > mmzone.h > lruvec

```
228  struct lruvec {
229      struct list_head        lists[NR_LRU_LISTS];
230      struct zone_reclaim_stat    reclaim_stat;
231      /* Evictions & activations on the inactive file list */
232      atomic_long_t           inactive_age;
233      /* Refaults at the time of last reclaim cycle */
```

include > linux > mmzone.h > lru_list > NR_LRU_LISTS

```
192  enum lru_list {
193      LRU_INACTIVE_ANON = LRU_BASE,
194      LRU_ACTIVE_ANON = LRU_BASE + LRU_ACTIVE,
195      LRU_INACTIVE_FILE = LRU_BASE + LRU_FILE,
196      LRU_ACTIVE_FILE = LRU_BASE + LRU_FILE + LRU_ACTIVE,
197      LRU_UNEVICTABLE,
198      NR_LRU_LISTS
199  };
```

2、反向映射

回收页表映射的匿名页或文件页时，需要从页表中删除映射，内核需要知道物理页被映射到哪些进程的虚拟地址空间，需要实现物理页到虚拟页的反向映射。

页描述符当中和反向映射具体成员如下：

include > linux > C mm_types.h

```
40  struct page {
41      /* First double word block */
42      unsigned long flags;        /* Atomic flags, some possibly
43                                   * updated asynchronously */

45          struct address_space *mapping;  /* If low bit clear, points to
59          pgoff_t index;      /* Our offset within mapping. */
89                  atomic_t _mapcount;
```

匿名页的反向映射视图如下：

include > linux > C mm_types.h > vm_area_struct

```
283  */
284  struct vm_area_struct {
285      /* The first cache line has the info for VMA tree walking. */
286
325      struct list_head anon_vma_chain; /* Serialized by mmap_sem &
```

include > linux > C rmap.h > anon_vma

```
28   struct anon_vma {
29       struct anon_vma *root;        /* Root of this anon_vma tree */
30       struct rw_semaphore rwsem;    /* W: modification, R: walking the l
58       struct rb_root rb_root; /* Interval tree of private "related" vmas */
```

include > linux > C rbtree.h > rb_root

```
42
43   struct rb_root {
44       struct rb_node *rb_node;
45   };
```

二、发起页回收

申请分配页的时候，页分配器首先尝试使用低水线分配页，如果使用低水线分配失败，说明内存轻微不足，页分配器将会唤醒所有符合分配条件的内存节点的页回收线程，异步回收页，然后尝试使用最低水线分配页。如果分配失败，说明内存严重不足，页分配器将会直接回收页。如果直接回收页失败，那么判断是否应该重新尝试回收页。

发起页回收源码分析如下：

mm > C page_alloc.c > __alloc_pages_nodemask(gfp_t, unsigned int, zonelist *, nodemask_t *)

```
4004    __alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
4005                           struct zonelist *zonelist, nodemask_t *nodemask)
4006    {
4007        struct page *page;
4008        unsigned int alloc_flags = ALLOC_WMARK_LOW;
```

1、异步回收

每个内存节点有一个页回收线程，如果内存节点的所有内存区域的空闲页数小于高水线，页回收线程就反复尝试回收页，调用函数shrink_node以回收内存节点中的页。

2、直接回收

针对备用区域列表中符合分配条件的每个内存区域，调用函数shrink_node来回收内存区域所属的内存节点中的页。回收页是以内存节点为单位执行的，函数shrink_node负责回收内存节点中的页。

3、判断是否应该重试回收页

函数should_reclaim_retry判断是否应该重试回收页，如果直接回收16次全都失败，或者即使回收所有可回收的页，也还是无法满足水线，则应该放弃重试回收。

mm > C page_alloc.c > ...

```
3585
3586  static inline bool
3587  should_reclaim_retry(gfp_t gfp_mask, unsigned order,
3588                       struct alloc_context *ac, int alloc_flags,
3589                       bool did_some_progress, int *no_progress_loops)
3590  {
3591      struct zone *zone;
3592      struct zoneref *z;
3593
```

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