Buddy和CMA简介
以及在Android中实际使用
CMA遇到问题的改进

2014/10/19

朱辉 zhuhui@xiaomi.com

CMA
UNMOVABLE
RECLAIMABLE
MOVABLE
RESERVE

目录

- Buddy系统结构简介
- CMA结构简介
- CMA的初始化
- CMA在普通内存结构中的分配和释放
- CMA对连续内存的分配
- CMA对连续内存的释放
- CMA在实际使用中遇到的问题和解决思路

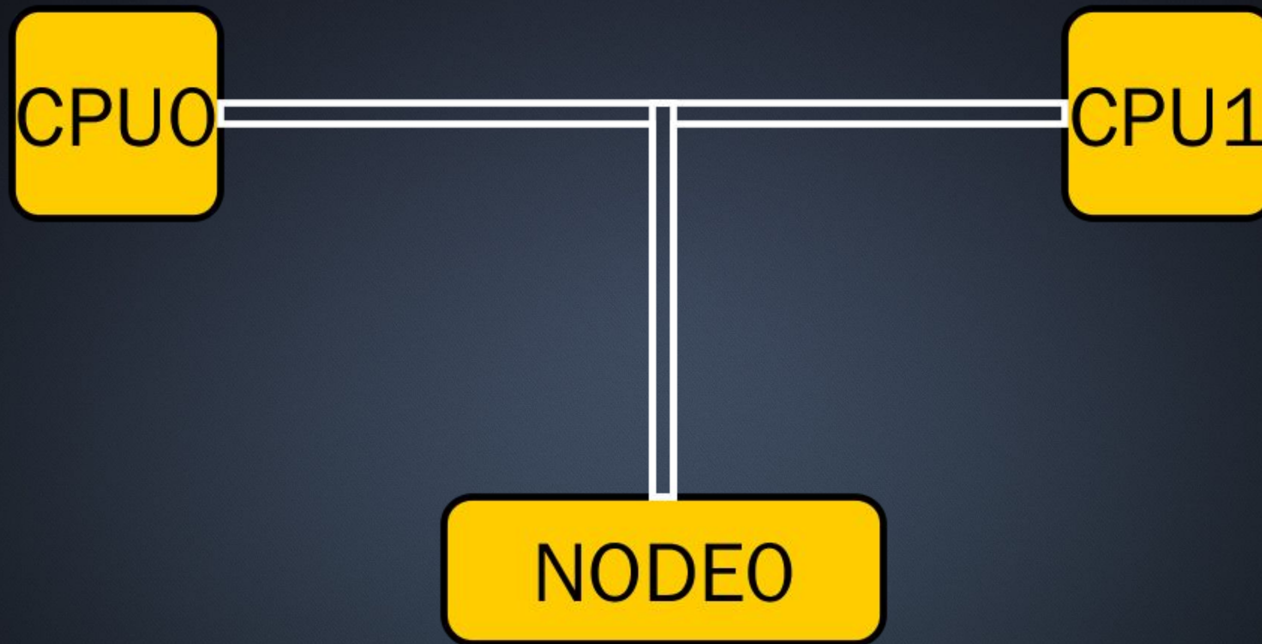
Buddy系统结构简介及内存分配和释放过程

- 介绍CMA从内存结构讲起因为其中有着千丝万缕的联系。
- 尤其是CMA本身就是Buddy系统的一部分。

均匀访存模型 UMA

- 从内存节点将起，讲内存节点不能不讲这两种访存模型。
- 每个CPU通过北桥中的内存控制器访问内存。
- 北桥易成为访问内存的瓶颈。
- 每个CPU访问内存速度一样。
- Linux内核把全部内存当成一个节点NODE。

均匀访存模型 UMA

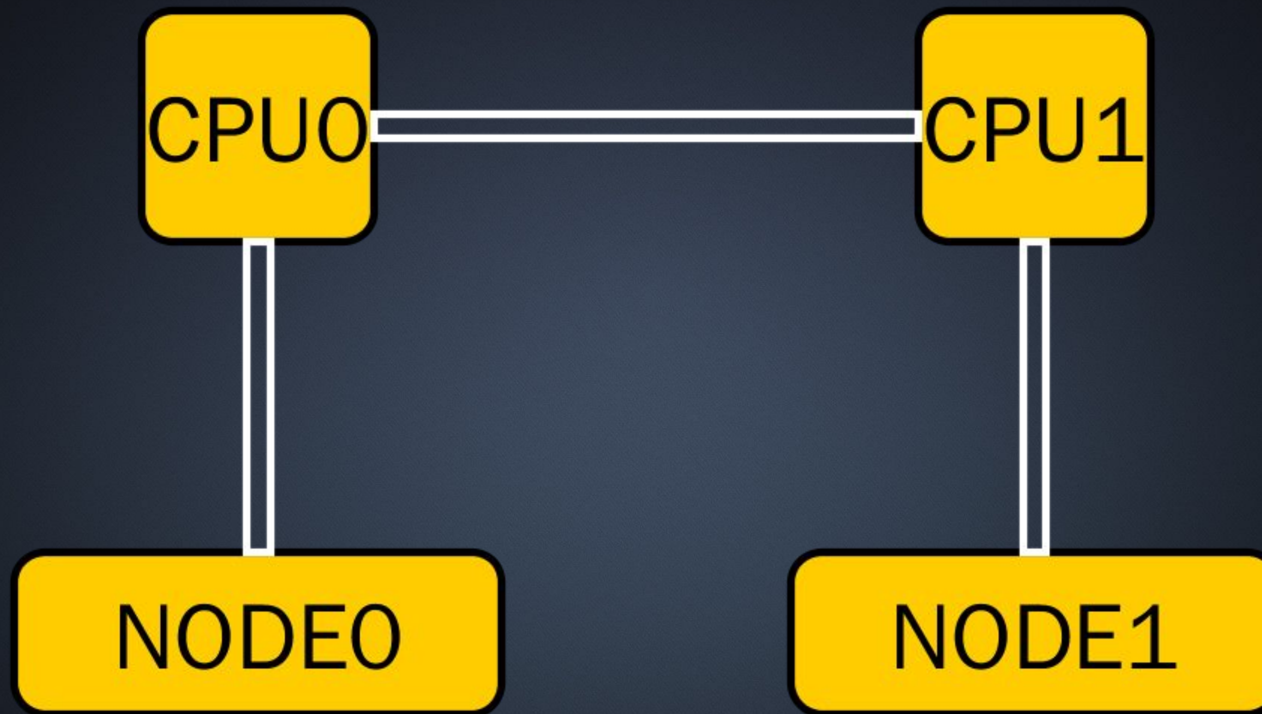


非均匀访存模型 NUMA

- 前一种模型的改进。
- 每个CPU自带内存控制器访问一部分内存。
- 内存相对每个CPU有本地内存和远端内存，访问速度不同。
- Linux内核将内存分成了若干节点NODE。
- NUMA的cache同步效率问题最近被扯出来狂黑。

- 内存分配的第一步基本就是对NODE的选择，系统一般会选择本地NODE。

非均匀访存模型 NUMA



节点（NODE）中的区域（ZONE）

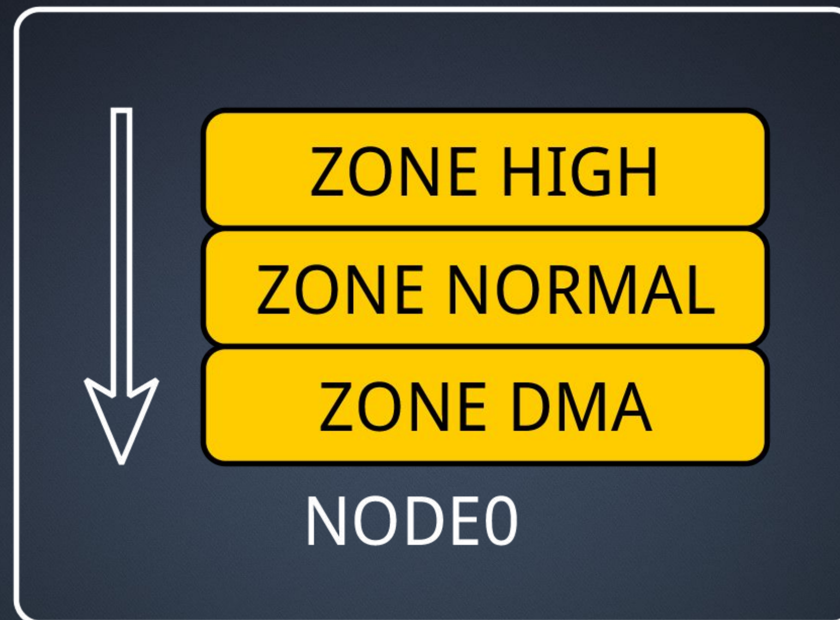
- 因为不同地址的内存情况不同，所以节点中的内存被分成了不同的区域。
- 不同的系统，ZONE的类型也不同，基本有以下几种：

  ZONE_DMA：可用作DMA的内存区域。

  ZONE_NORMAL：直接被内核直接映射到自己的虚拟地址空间的地址。

  ZONE_HIGHMEM：不能被直接映射到内核的虚拟地址空间的地址。
- 基本上就是越接近0的内存越珍贵，所以在内存分配时是先选择起始ZONE，然后从高到低试图从其中分配内存。

节点（NODE）中的区域（ZONE）



早期区域（ZONE）中的结构(Buddy)

- Buddy是为了减少内存碎片而设计。
- ZONE中大小为MAX_ORDER的数组，数组中每一个元素是一个列表，其依次索引了2的0次方到2的(MAX_ORDER-1)次方大小的页，这里索引的就是空闲页。
- 关于PCP，当释放order为0的页的时候，会先分配若干页order为0的页到PCP中，PCP中页超过一定量才会真的释放，。

  分配order为0的页也是先从PCP从分配。
- 分配时，如果order是0，先试图从PCP中分配页，如果不是，根据分配指定的order到相应的数组order项目中的列表去取页大小为order的页返回。

  如果没有就到order+1列表中取出1页并分为2页，这样每页就是order大小，1页返回，另1页加入order列表。

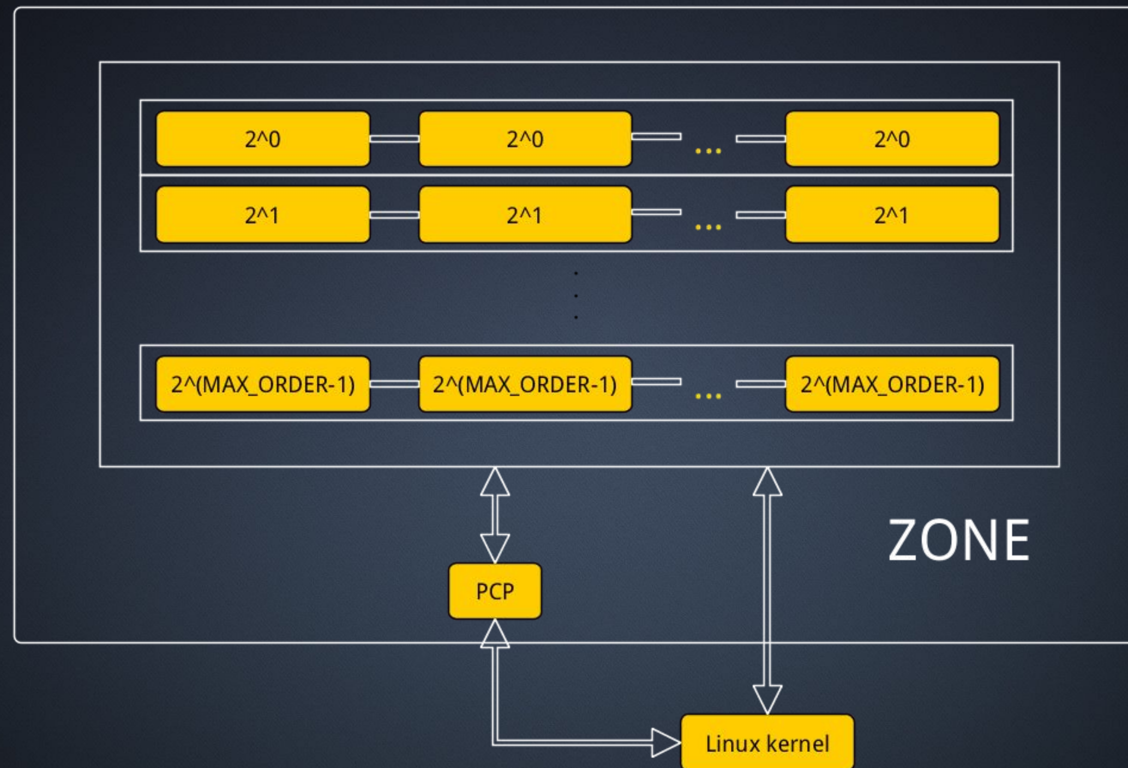
  如果order+1列表中也没有页，则到order+2列表中去拆页。

  后面依此类推。
- 释放页时，在order列表中找buddy，如果没找到加回order列表。

  如果找到，则合并为一个order+1页大小的页并加回order+1列表。

  后面依此类推。
- Buddy的问题是所有减少碎片的操作，都在释放才做，如果一个页持续不释放，一个页块中其他页都会被影响。所以在2.6.24开始对这个设计进行了扩展。

早期区域（ZONE）中的结构(Buddy)



现在区域（ZONE）中的结构(Buddy+Migrate)

- 将ZONE中大小为MAX_ORDER的数组中的每一个列表替换为一个MIGRATE_TYPES大小的数组，而数组中每一个元素是一个列表，其依次索引了2的0次方到2的(MAX_ORDER-1)次方大小的页。

  因为实际使用中，每个Migrate块内部更加紧密，所以把每个Migrate块画在了一起，和实际实现不同。

- 下面是几种迁移块的介绍：

  MIGRATE_UNMOVABLE，在内存当中有固定的位置，不能移动。内核自己多会分配这种类型的数据。

  MIGRATE_RECLAIMABLE，不能直接移动，但可以删除，其内容页可以从其他地方重新生成。例如，映射自文件的数据属于这种类型，针对这种页，内核有专门的页面回收处理。

  MIGRATE_MOVABLE，可以随意移动，用户空间应用程序所用到的页属于该类别。它们通过页表来映射，如果他们复制到新的位置，页表项也会相应的更新，应用程序不会注意到任何改变。

  MIGRATE_RESERVE，保留页。

  MIGRATE_ISOLATE，用来帮助做页的迁移，这里索引的页不能分配。

- 在物理页层次上，每个2的MAX_ORDER-1次方大小的页，也就是Buddy最大的一层的页，被规定为一个页块，页块上会记录其的迁移类型。

现在区域（ZONE）中的结构(Buddy+Migrate)

- 在请求页时，内核已经选择了要从哪个migrate中分配。具体相关函数allocflags_to_migratetype。
- 分配时首先在相关migrate块中进行，过程和早期方法一样，层层找页。相关函数__rmqueue_smallest。
- 如果在当前migrate块中找不到页，就按照后备规则（其中MIGRATE_RESERVE并不是一项，而是标明这条规则的结束）进行横向并从下向上的找页。

  也就是从MAX_ORDER-1开始，把规则中每个migrate块过一下，如果没有再进入到MAX_ORDER-2进行每个migrate块的找页。

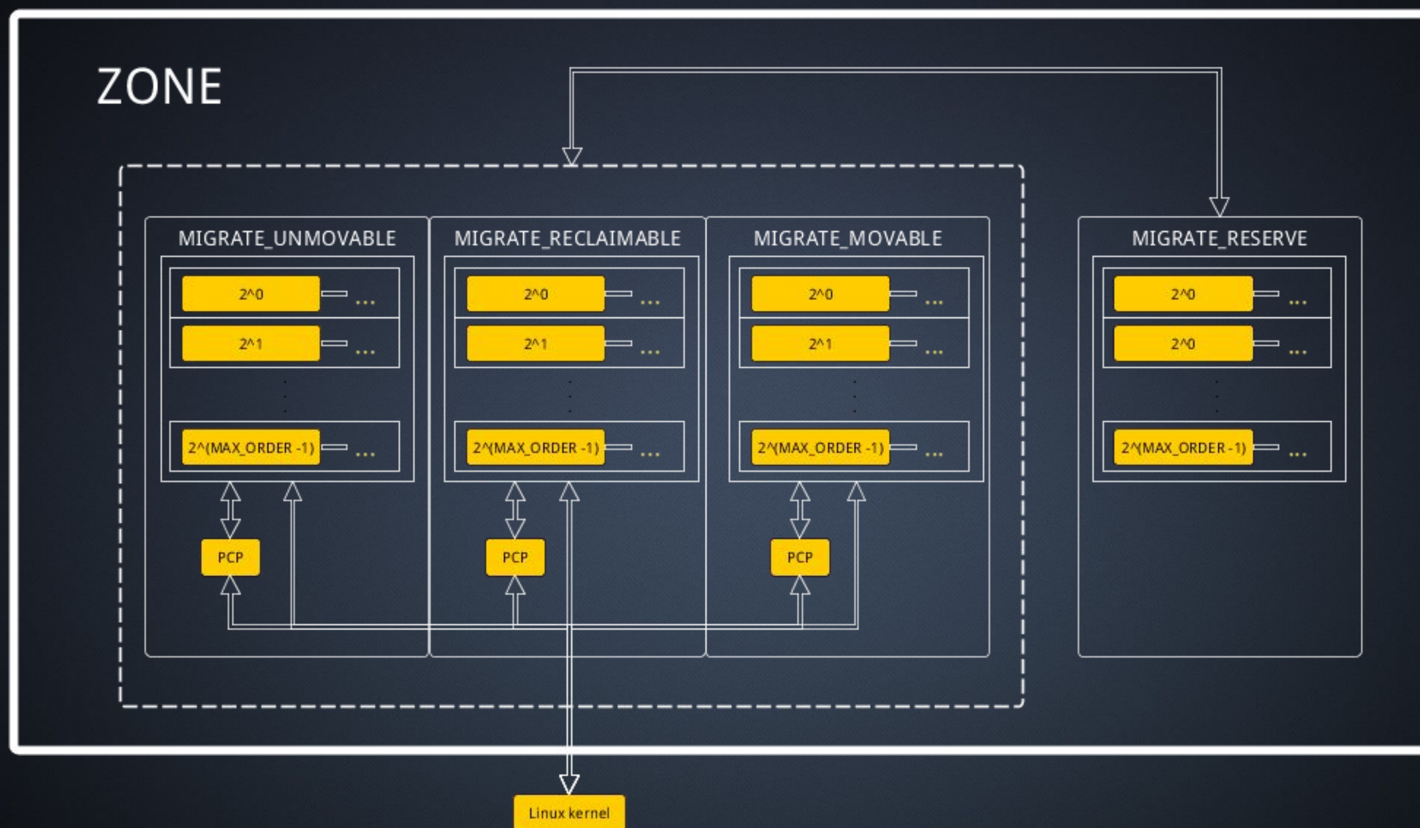
  在找到合适的页的时候，还会根据条件做页的迁徙，把分配出的页的整个页块迁移到当前要分配的migrate块上（页块上的记录也会被改变）。相关函数try_to_steal_freepages。这里的用词有点个诡异，为什么不直接用migrate?

  相关函数__rmqueue_fallback。
- 如果还没找到，就到MIGRATE_RESERVE以从上到下的普通方式找页。如果还没有，就失败。
- MIGRATE_UNMOVABLE，MIGRATE_RECLAIMABLE，MIGRATE_MOVABLE是可互相迁移内存的，所以把他们画在一起，而MIGRATE_RESERVE是不能的，所以被分开。
- 从对迁移块的使用可以看到，除了反向找页和根据需求作页迁徙这种效率的设计以外，迁移块中MIGRATE_MOVABLE和MIGRATE_RECLAIMABLE可以通过换页和释放页来解决Buddy面临的碎页问题（这里我没找到实际代码，不过在思路上是可行的），而MIGRATE_UNMOVABLE可以通过到这两个迁移块中找页解决自己的无大页问题。

现在区域（ZONE）中的结构(Buddy+Migrate)

- 页的释放和传统方式一样，增加的部分是回到哪个迁移块要由页块中的migrate类型来决定。
- 从这里我们可以看到在现在的内存结构中，页块上的迁移类型非常重要，标记了每个页“回家的路”。

现在区域（ZONE）中的结构(Buddy+Migrate)



后备（fallbacks）规则

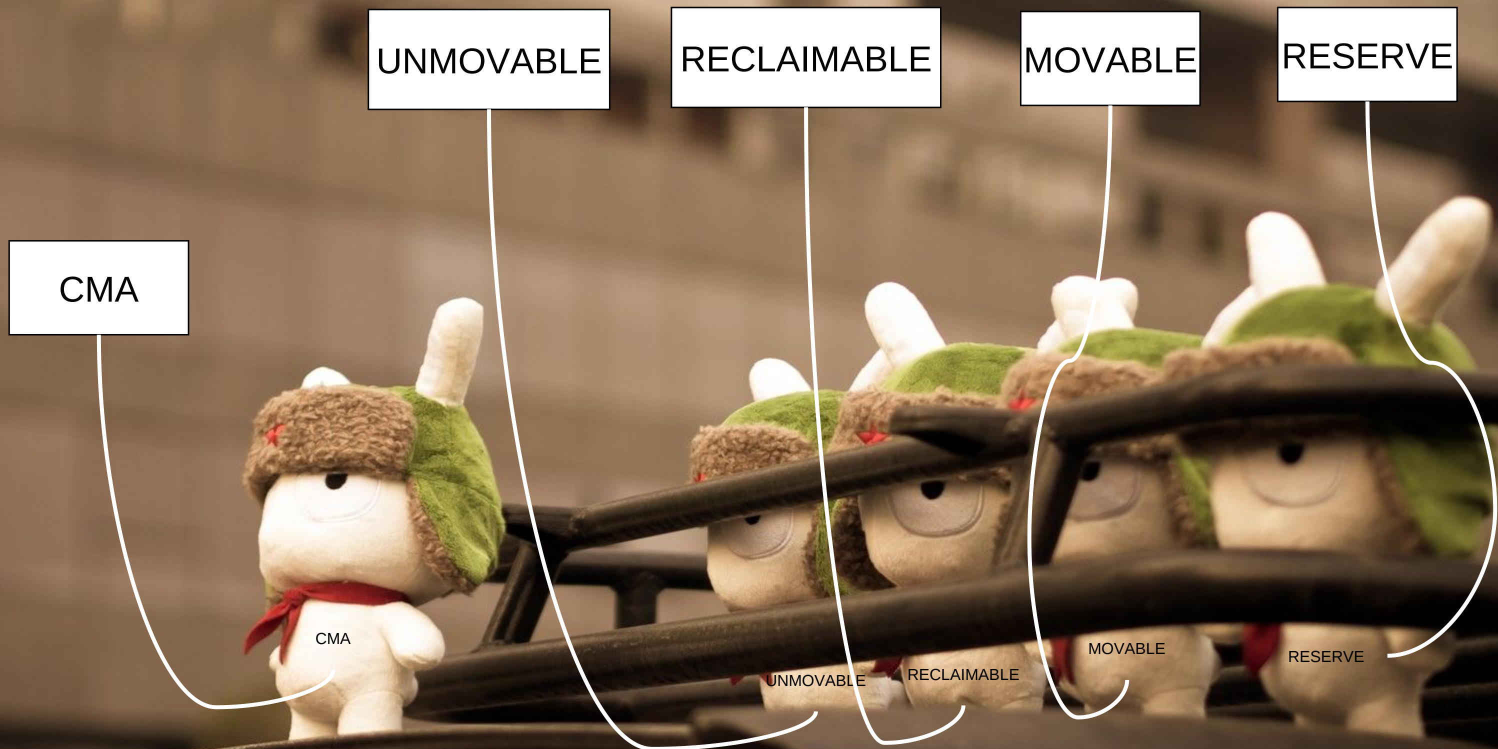
```
static int fallbacks[MIGRATE_TYPES][4] = {
    [MIGRATE_UNMOVABLE]   = { MIGRATE_RECLAIMABLE, MIGRATE_MOVABLE,     MIGRATE_RESERVE },
    [MIGRATE_RECLAIMABLE] = { MIGRATE_UNMOVABLE,   MIGRATE_MOVABLE,     MIGRATE_RESERVE },
#ifdef CONFIG_CMA
    [MIGRATE_MOVABLE]     = { MIGRATE_CMA,         MIGRATE_RECLAIMABLE, MIGRATE_UNMOVABLE, MIGRATE_RESERVE },
    [MIGRATE_CMA]         = { MIGRATE_RESERVE }, /* Never used */
#else
    [MIGRATE_MOVABLE]     = { MIGRATE_RECLAIMABLE, MIGRATE_UNMOVABLE,   MIGRATE_RESERVE },
#endif
    [MIGRATE_RESERVE]     = { MIGRATE_RESERVE }, /* Never used */
#ifdef CONFIG_MEMORY_ISOLATION
    [MIGRATE_ISOLATE]     = { MIGRATE_RESERVE }, /* Never used */
#endif
};
```

现有内存结构的限制

- 无法分配超过2的(MAX_ORDER-1)次方大小的连续页。
- 即使要分配的页小于2的(MAX_ORDER-1)次方大小，也不能保证某个区域(ZONE)页有这个页。
- 以前有采用alloc_bootmem在启动建立Buddy之前分配内存，但是这些内存无法再被Buddy系统使用，造成了内存的浪费。
- 所以内核于2012年5月引入CMA，或者可称为Contiguous Memory Allocator，连续内存分配器。

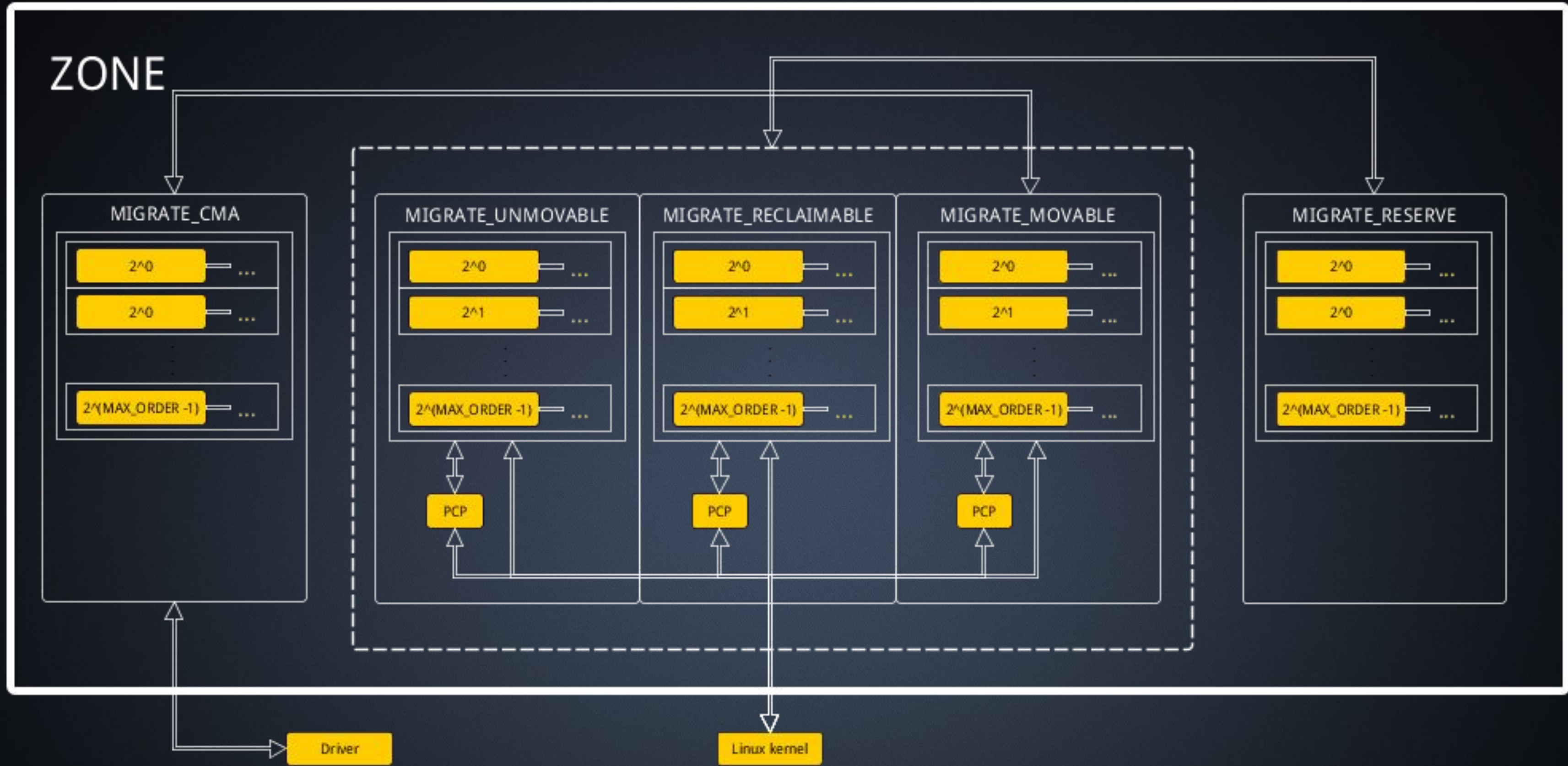
CMA结构简介

- 请找彩蛋


CMA是一个特殊的Migrate类型
UNMOVABLE
RECLAIMABLE
MOVABLE
RESERVE
CMA
CMA
UNMOVABLE
RECLAIMABLE
MOVABLE
RESERVE
19

CMA结构简介

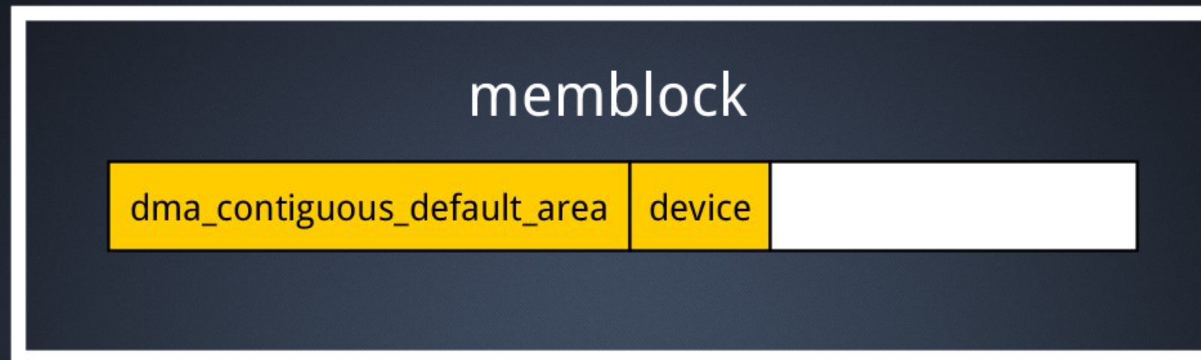
- MIGRATE_CMA也是一个迁移类型。
- CMA用来解决前面提到的问题。
- 后面介绍分配会详细的介绍这张图。


ZONE
MIGRATE_CMA
2^0
2^0
2^(MAX_ORDER -1)
MIGRATE_UNMOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RECLAIMABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_MOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RESERVE
2^0
2^0
2^(MAX_ORDER -1)
Driver
Linux kernel

CMA的初始化过程

- 在系统初始化还没将内存交给Buddy系统之前，用dma_contiguous_reserve_area函数可以帮助驱动或者系统申请memblock中的一块区域为CMA，并添加到cma_areas数组中。
- 系统默认申请的申请的存入dma_contiguous_def_base。
- 驱动可以根据需要申请自己的CMA区域。

dma_contiguous_reserve_area

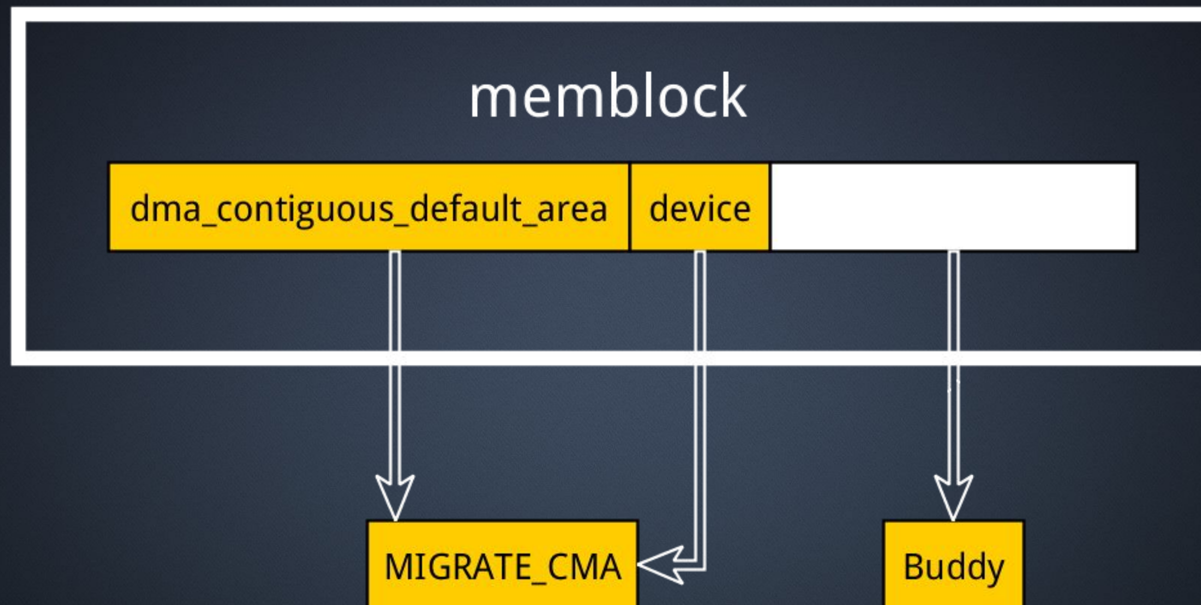


Buddy

CMA的初始化过程

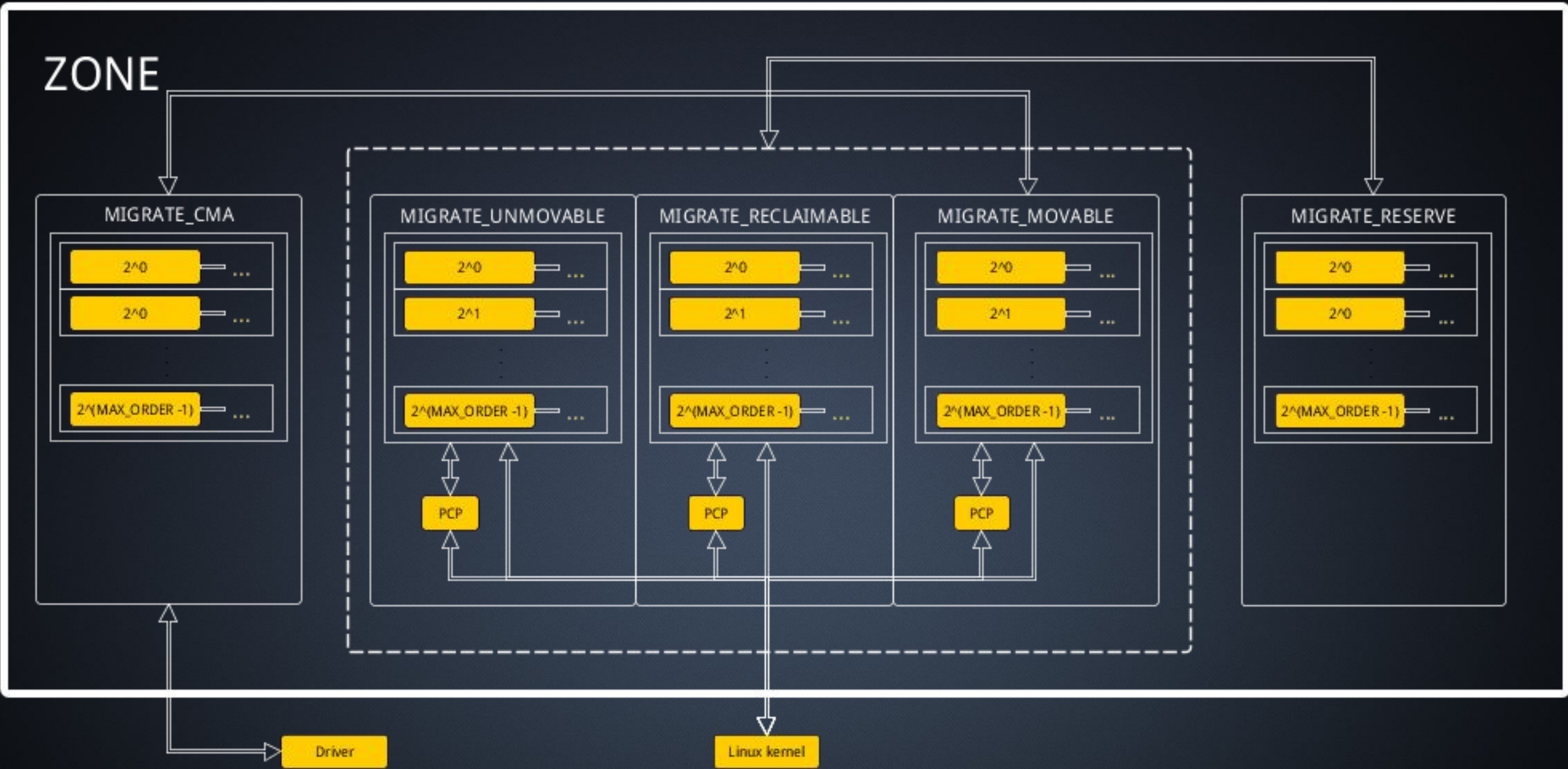
- 在其他内存都被放到Buddy系统后，因为前面已经分配出去，所以CMA的页并没有被初始化到Buddy系统中。
- 这时系统会调用函数cma_init_reserved_areas，其会对CMA结构进行最后的初始化（因为内存相关的操作都要在Buddy系统有内存后才能做，例如分配bitmap）。然后将CMA结构中的每个页都存入MIGRATE_CMA块，并将全部页块标记为MIGRATE_CMA。
- CMA初始化完成，从这个初始化就可以看到CMA内存分别被cma_areas数组和Buddy系统索引。

cma_init_reserved_areas



CMA在Buddy内存结构中的分配和释放

- 根据后备规则，MIGRATE_CMA是MIGRATE_MOVABLE的第一个后备块。
- 其分配方式和其他项目一样，唯一的区别是在前面介绍过的函数try_to_steal_freepages中，MIGRATE_CMA在任何情况下都不能被迁移到其他迁移块。
- 在释放时，这些页也会回到MIGRATE_CMA块中。


ZONE
MIGRATE_CMA
2^0
2^0
2^(MAX_ORDER -1)
MIGRATE_UNMOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RECLAIMABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_MOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RESERVE
2^0
2^0
2^(MAX_ORDER -1)
Driver
Linux kernel

后备（fallbacks）规则

```
static int fallbacks[MIGRATE_TYPES][4] = {
    [MIGRATE_UNMOVABLE]   = { MIGRATE_RECLAIMABLE, MIGRATE_MOVABLE,     MIGRATE_RESERVE },
    [MIGRATE_RECLAIMABLE] = { MIGRATE_UNMOVABLE,   MIGRATE_MOVABLE,     MIGRATE_RESERVE },
#ifdef CONFIG_CMA
    [MIGRATE_MOVABLE]     = { MIGRATE_CMA,         MIGRATE_RECLAIMABLE, MIGRATE_UNMOVABLE, MIGRATE_RESERVE },
    [MIGRATE_CMA]         = { MIGRATE_RESERVE }, /* Never used */
#else
    [MIGRATE_MOVABLE]     = { MIGRATE_RECLAIMABLE, MIGRATE_UNMOVABLE,   MIGRATE_RESERVE },
#endif
    [MIGRATE_RESERVE]     = { MIGRATE_RESERVE }, /* Never used */
#ifdef CONFIG_MEMORY_ISOLATION
    [MIGRATE_ISOLATE]     = { MIGRATE_RESERVE }, /* Never used */
#endif
};
```

小问题

后面要介绍CMA对连续内存的分配和释放，是比较枯燥的一段，所以我在这里提两个小问题增加点乐趣，能在我自己解答前回答一个问题的同学，送F码一个：

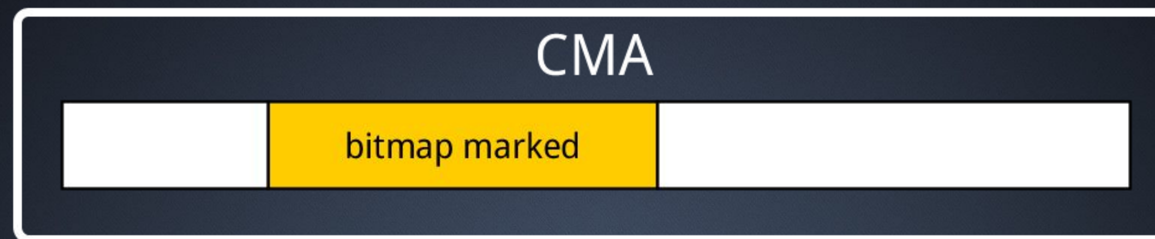
1. 这个问题和刚才介绍的部分相关，不想听后面话题的同学可以思考这个，CMA在普通内存结构中的分配是有一定问题的，这也是我在这个话题最后要介绍的，这个问题是什么？

2. 这个问题和后面介绍比较相关，为什么MIGRATE_CMA不能迁移给MIGRATE_MOVABLE？

CMA对连续内存的分配

- 首先通过驱动结构dev，找到要使用的cma结构，如果找不到则使用系统分配的dma_contiguous_default_area。
- 进入cma_alloc函数，先对CMA结构加锁，通过CMA结构中的bitmap找到合适的数据块起始位置并做标记。
- 解锁CMA结构。

CMA对连续内存的分配和释放



MIGRATE_ISOLATE free list

alloc_contig_range

- 函数cma_alloc将加全局锁cma_mutex。
- 然后调用函数alloc_contig_range真正的对这块CMA内存进行分配。
- 函数alloc_contig_range返回后会对cma_mutex解锁。

- 这个锁的范围在CMA加入后有不少变化，解决了一些死锁的问题。实际使用的早期内核版本的时候要小心死锁的问题。

- 下面对alloc_contig_range内部过程进行介绍。

start_isolate_page_range

- 函数alloc_contig_range首先会调用start_isolate_page_range。
- 先将全部页块标记为MIGRATE_ISOLATE，这样后面这些页被自动放入MIGRATE_ISOLATE的空闲列表。

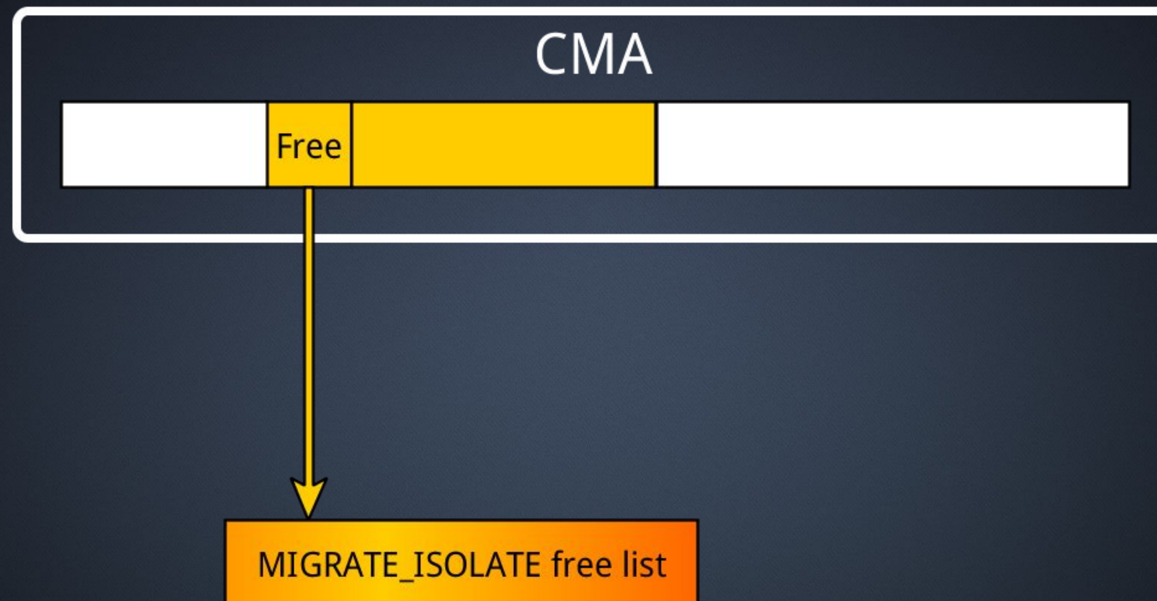
start_isolate_page_range

MIGRATE_ISOLATE free list

start_isolate_page_range

- 这是已经在CMA空闲列表中的页反而有可能被分配出去，而且没有操作能自动将他们移到MIGRATE_ISOLATE空闲列表中，所以讲这些空闲页移动到MIGRATE_ISOLATE空闲列表中。
- 接着这个函数就返回了。

start_isolate_page_range



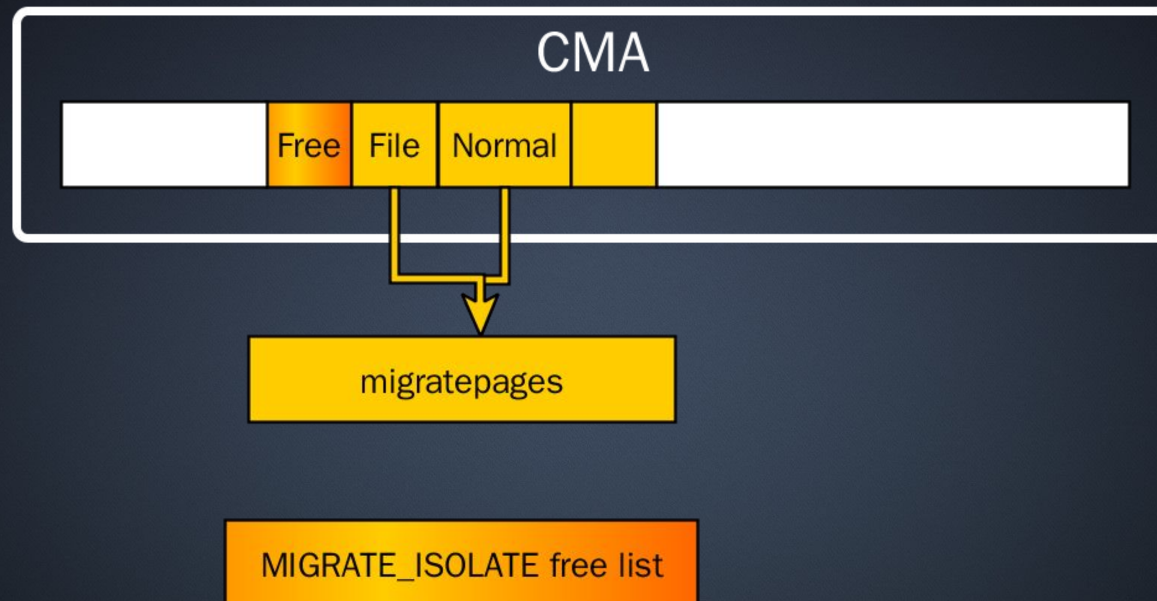
__alloc_contig_migrate_range

- 接alloc_contig_range函数会调用函数__alloc_contig_migrate_range，这个函数的目的就是把剩下的页都移到MIGRATE_ISOLATE空闲列表中。
- 其首先会调用函数migrate_prep，其会调用函数lru_add_drain_all。这个函数的主要作用是把lru列表中的页都删掉，方便后续处理。
- 接着其会循环调用函数isolate_migratepages_range、函数reclaim_clean_pages_from_list、函数migrate_pages，对全部非空闲的页进行处理。
- 对所有页处理完成后，这个函数就会返回。
- 下面对这三个函数依次进行介绍。

isolate_migratepages_range

- 找出没处理过的COMPACT_CLUSTER_MAX个页，放入一个列表cc->migratepages。

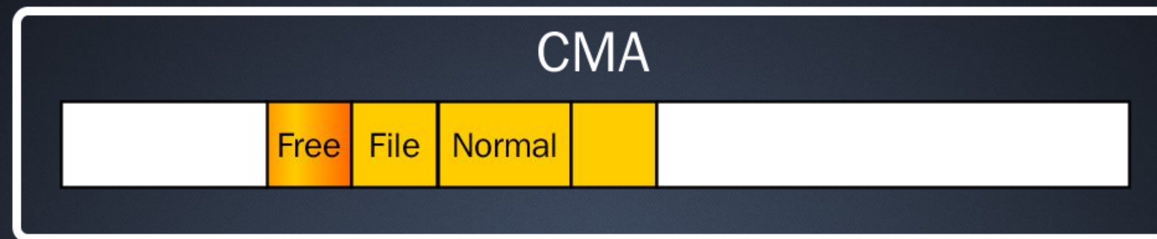
isolate_migratepages_range

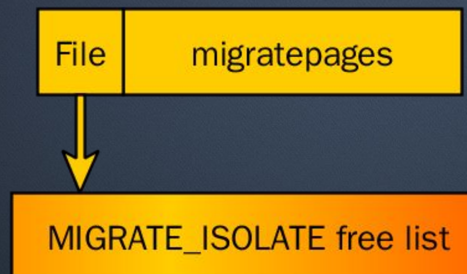


reclaim_clean_pages_from_list

- 将cc->migratepages中干净可以回收的页，一般是文件cache，回收并释放掉。
- 因为页块已经被标记为MIGRATE_ISOLATE，所以其会自动被加入到MIGRATE_ISOLATE空闲页中。

reclaim_clean_pages_from_list





第二个小问题是否有人回答

- 后面就会揭晓答案

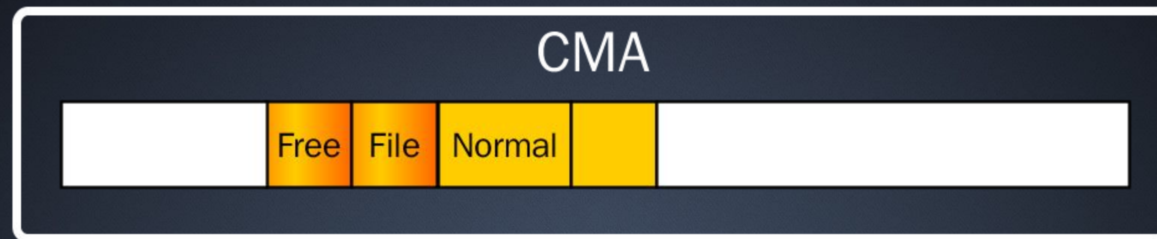
migrate_pages

- 对无法通过其他方式释放掉的页，就会到这里进行最终的迁移页处理。
- 这个函数会先分配一个页，两个页进行切换，最后将页释放掉并交给MIGRATE_ISOLATE空闲页。
- 前面问题2答案也揭晓了，CMA只能作为MIGRATE_MOVABLE分配出去，因为只有MIGRATE_MOVABLE类型是随便移动的。

  但是如果CMA迁移成MIGRATE_MOVABLE，因为MIGRATE_UNMOVABLE，MIGRATE_RECLAIMABLE和MIGRATE_MOVABLE可能互相迁徙，所以CMA就有可能迁移给另两者，则无法再移动。

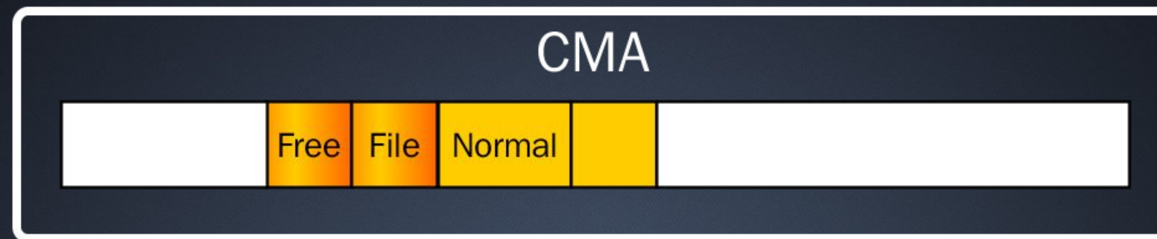
  则CMA就无法再当连续页分配了。

migrate_pages



MIGRATE_ISOLATE free list

migrate_pages



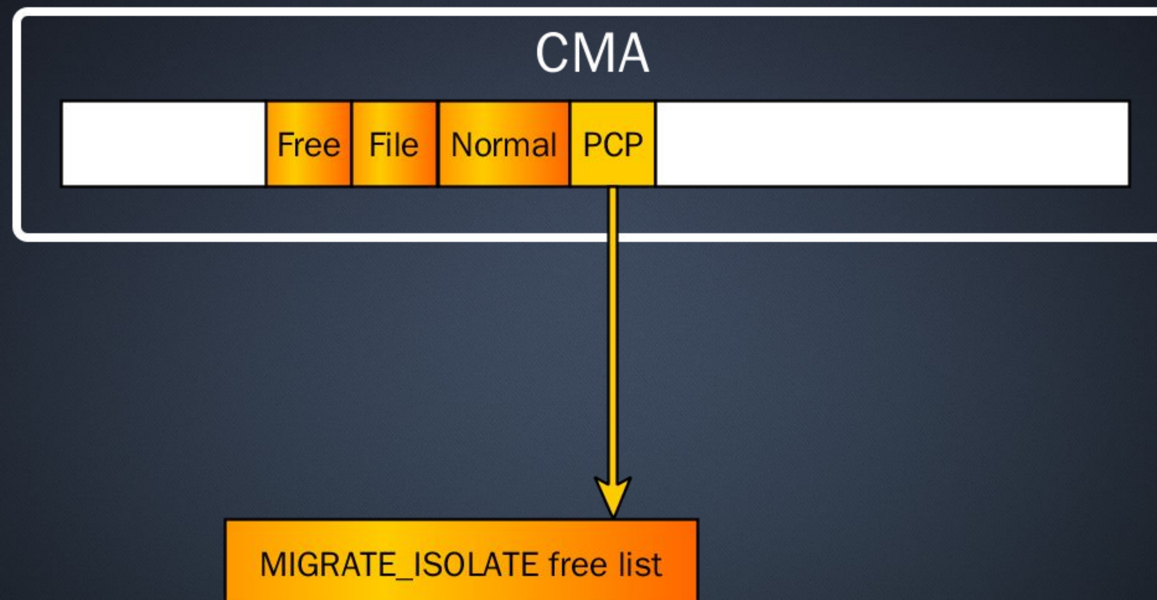
migratepages(Normal)

MIGRATE_ISOLATE free list

drain_all_pages

- 从函数__alloc_contig_migrate_range返回后，alloc_contig_range函数会再次调用lru_add_drain_all()，这里为何再次调用没搞明白。
- 接着调用drain_all_pages()，将PCP中缓冲的页都释放掉，从而其中标记为MIGRATE_ISOLATE的页会被释放倒MIGRATE_ISOLATE空闲列表中。

drain_all_pages



页面分配前的扩张

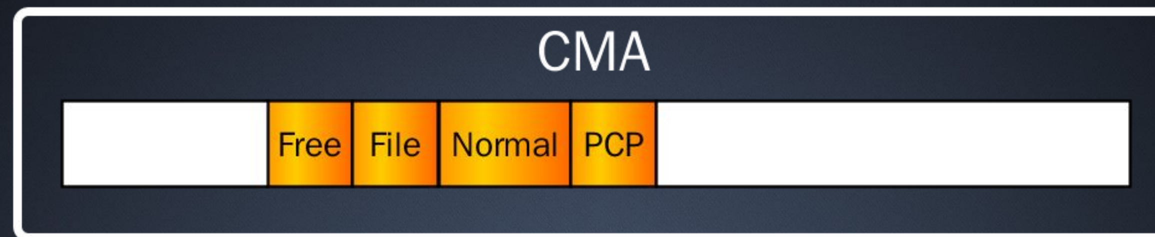
- 现在全部页面都已经都进入了MIGRATE_ISOLATE空闲列表中了，唯一的问题是开头的结尾的部分页可能在Buddy结构中一个大页中的一部分。

  函数alloc_contig_range对他们的解决办法对开头页进行检查，如果是则将分配的开头地址向这个块的开头页扩展。
- 后续操作在页面分配结束后介绍。

test_pages_isolated

- 这里对全部页面的状态进行最后的检查。如果有状态不对的，函数会出错返回。

test_pages_isolated

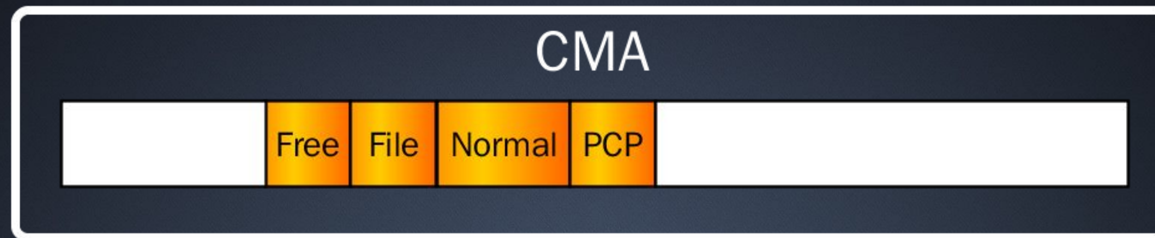


MIGRATE_ISOLATE free list

isolate_freepages_range

- 这里将指定好的页面从Buddy系统中抽取出来标记为已经使用。

isolate_freepages_range



MIGRATE_ISOLATE alloced

页面分配后的收缩

- 将刚才多分配的页面全部释放掉，通过这两步的操作，分配给连续内存的页面就不再是某个Buddy块中的一部分。

undo_isolate_page_range

- 最后函数alloc_contig_range会将全部标记为MIGRATE_ISOLATE的页面标记回MIGRATE_CMA。

  因为这时要分配给连续内存的页已经都从Buddy系统中抽出去了，不论任何状态都不再会被使用到了。

- 函数alloc_contig_range返回，函数cma_alloc返回，最终连续页被返回给驱动。

undo_isolate_page_range

CMA对连续内存的释放

- 因为分配出去的页已经被标记为MIGRATE_CMA了，所以对他们的释放就很简单。
- 先把这些页释放掉，让他们回到MIGRATE_CMA空闲列表。
- 接着清楚掉bitmap中的项目，就可以了。

CMA对连续内存的释放

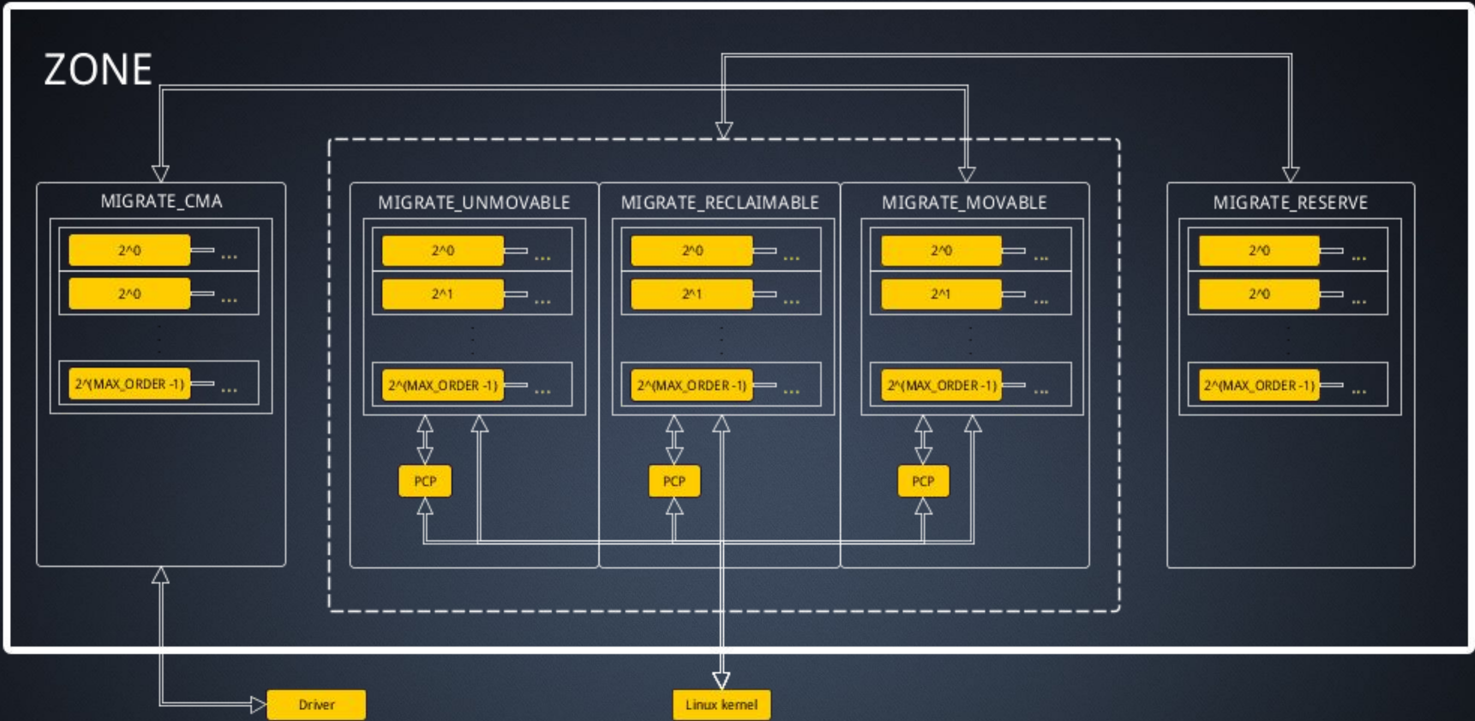
现在要回答问题1了

- 有没有同学回答?

CMA在实际使用中遇到的问题和解决思路

- 因为MIGRATE_CMA只是MIGRATE_MOVABLE的备份项目。

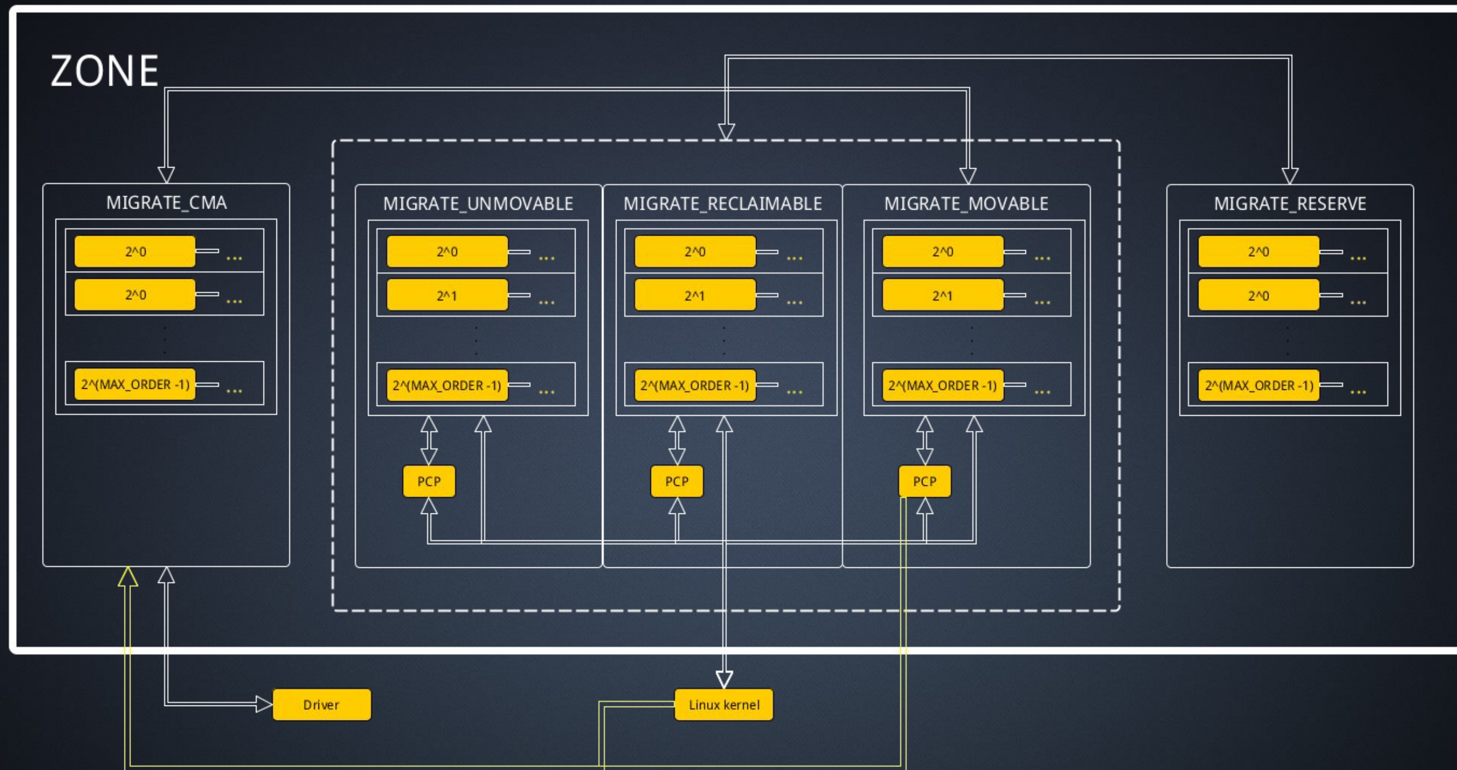
  在系统初始化时，普通内存除了MIGRATE_RESERVE，其他内存都是标记为MIGRATE_MOVABLE的。在很多系统包括Android系统，MIGRATE_MOVABLE块占用的内存也最大。
- 所以使用到MIGRATE_CMA时，其之外的内存已经所剩无几。而这时系统很容易除非lowmemorykiller，如果设置lowmemorykiller的规则，则可能触发的是oomkiller。
- 最终的结果是即使系统经常杀掉进程，CMA中还有大量内存可用，CMA没有起到原来的设计目的。


ZONE
MIGRATE_CMA
2^0
2^0
2^(MAX_ORDER -1)
MIGRATE_UNMOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RECLAIMABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_MOVABLE
2^0
2^1
2^(MAX_ORDER -1)
PCP
MIGRATE_RESERVE
2^0
2^0
2^(MAX_ORDER -1)
Driver
Linux kernel

CMA_AGGRESSIVE

- 增加了一个叫CMA_AGGRESSIVE的功能，其主要思路是在找页第一阶段，函数__rmqueue_smallest中，如果条件允许，先从CMA中找页。
- 现在引入了一个新的问题，这样大部分的CMA页都会被分出去，在分配连续页的时候会有一些问题被引入。

CMA_AGGRESSIVE



新引入问题的处理

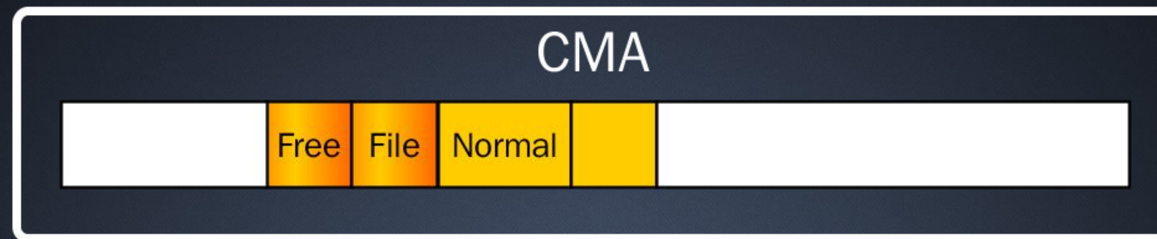
- 函数migrate_pages中，因为有换页的操作，这里就有个分配页的问题。如果系统中本身内存页比较少，就很可能需要调用函数shrink_slab取得一部分内存。但是因为系统结构的限制，这个接口每次只能释放COMPACT_CLUSTER_MAX个页，系统一般定义为32。

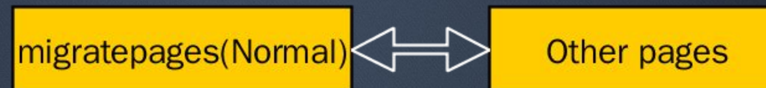
  而这里分配页也会使用到函数__rmqueue_smallest，很可能会再次去分配CMA的页。
- 所以当分配的连续内存比较多，而且系统内存本身已经不足，很可能整个分配连续内存的过程会变的缓慢并且出错率很高。

- 解决方法：

1. 在申请连续内存开始的部分根据系统内存大小调用一次shrink相关函数分配出一部分free内存，提高换页的成功率和速度。
2. 在申请连续页的开头和结尾操作原子变量作为轻量锁，在函数__rmqueue_smallest先去检查这个锁，成功才去从CMA中分配页，介绍Buddy系统分配页和CMA系统分配页的冲突。

migrate_pages





MIGRATE_ISOLATE free list

谢谢！问题？

- weibo: @teawater_z 欢迎在线吐槽