Linux物理内存描述

http://www.ilinuxkernel.com

目　　录

1 概述

Linux内核采用页式存储管理，进程的地址空间被划分成固定大小的“页面”；物理内存同样被分为与页面大小相同的“页帧”，由MMU在运行时将虚拟地址“映射”成某个物理内存页面上的地址。

本文以linux-2.6.32-220.el6版本内核源码为基础，介绍Linux内核中有关物理内存的概念，和如何描述物理内存。注意：本文中涉及到的仅是和物理内存有关的概念、数据结构。和地址如何映射、线性地址、物理地址；内存如何分配/回收等概念无关。

2 物理内存相关概念

2.1 NUMA（Non Uniform Memory Access）

在现代计算机中，CPU访问内存的速度与两者之间的距离有关，导致访问不同内存区域的时间可能不同。如下图，是系统中有2个CPU（可以超过2个CPU）时，NUMA内存访问模型。

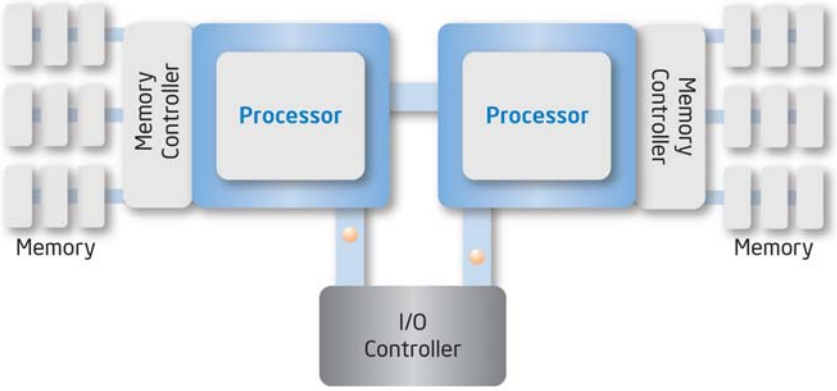

图1 NUMA内存访问模型

图2是Intel Nehalem（Xeon E5xxx系列）架构中NUMA内存访问模型。每个CPU下面都有物理内存，两个CPU之间通过QPI总线通信。黑色箭头表示访问本地CPU下面的物理内存，绿色箭头为访问另外CPU下的物理内存。本地CPU访问其他CPU下面内存时，数据要通过QPI总线传输。显然访问本地内存速度最快，访问其他CPU下面的内存速度较慢。这就是NUMA的由来

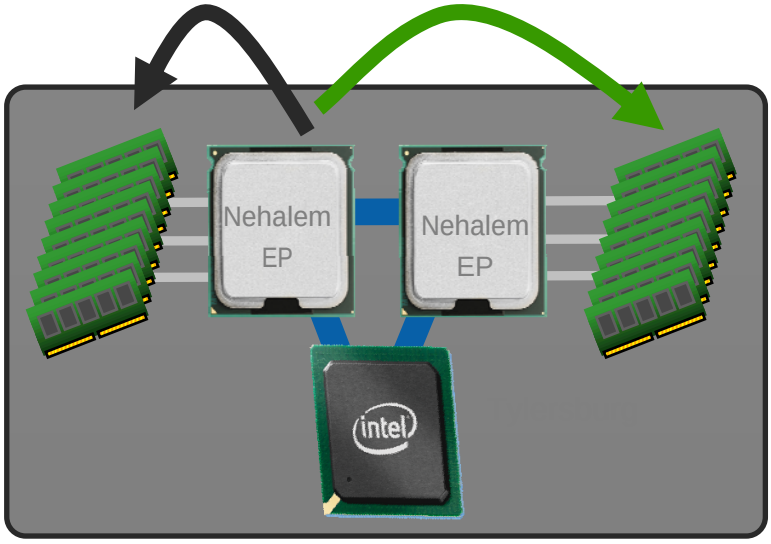

图2 Intel Nehalem架构中NUMA内存访问模型

下面两个图分别是Nehalem架构下，本地内存访问和远端内存访问模型。

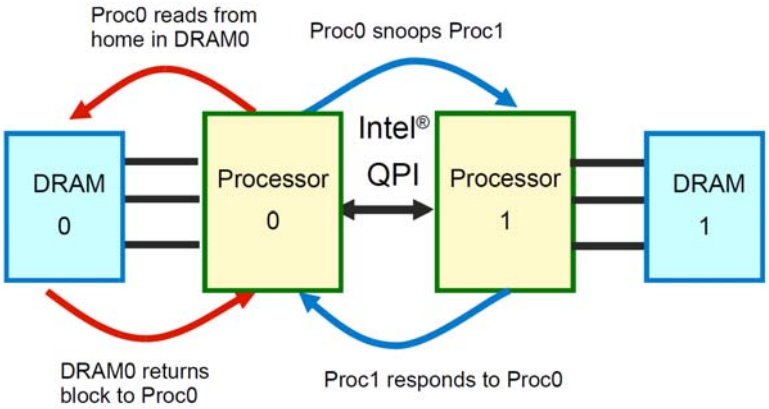

图3 Intel Nehalem架构中本地内存访问

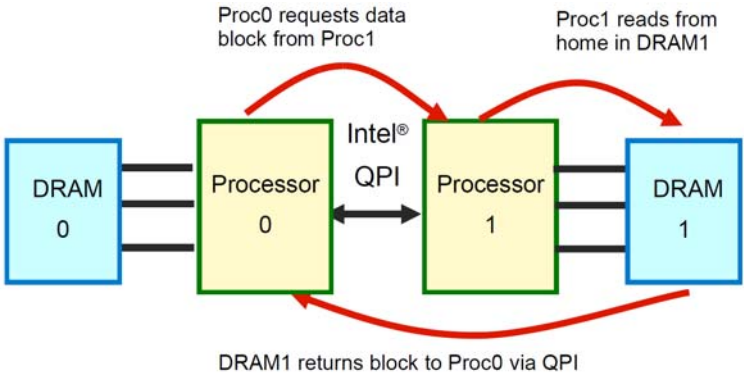

图4 Intel Nehalem架构中远端内存访问

2.2 页面和虚拟内存（Paging and Virtual Memory）

当打开Intel CPU页面机制时，线性地址空间就划分成页面，虚拟地址的页面然后映射到物理页面。当Linux使用页面机制时，对上层应用程序是透明的。即**Intel CPU就是这么实现的**，采用段页式机制，所有运行在Intel CPU上的OS都要遵守页面机制，包括Windows、Linux都是如此。

下图是x86/x86_64架构中，页面大小为4K时，页面属性。

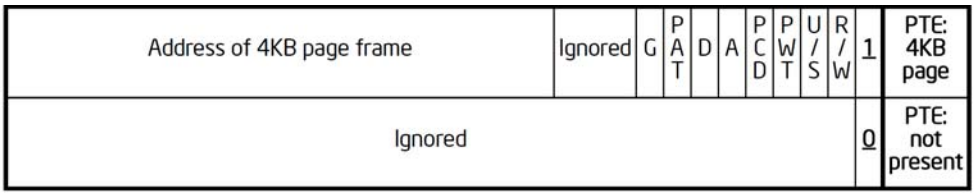

图5 x86中物理页面属性（Page Table Entry）

下面表为页面属性（Page Table Entry）各字段含义。这些字段包括线性地址对应的物理页面是否存在、页面是否可读写、是用户模式还是系统模式、页面是否为脏等。这些属性在Linux内核实现中都会用到。

表1 页面属性（Page Table Entry）各字段含义

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

为了便于理解页面属性的作用，我们来看一个示例。如系统有两个进程i和j，内核可以设置该进程使用的物理页面属性，如进程对某些页面只能读，而不能写；对某些页面可读可写。这样既可以

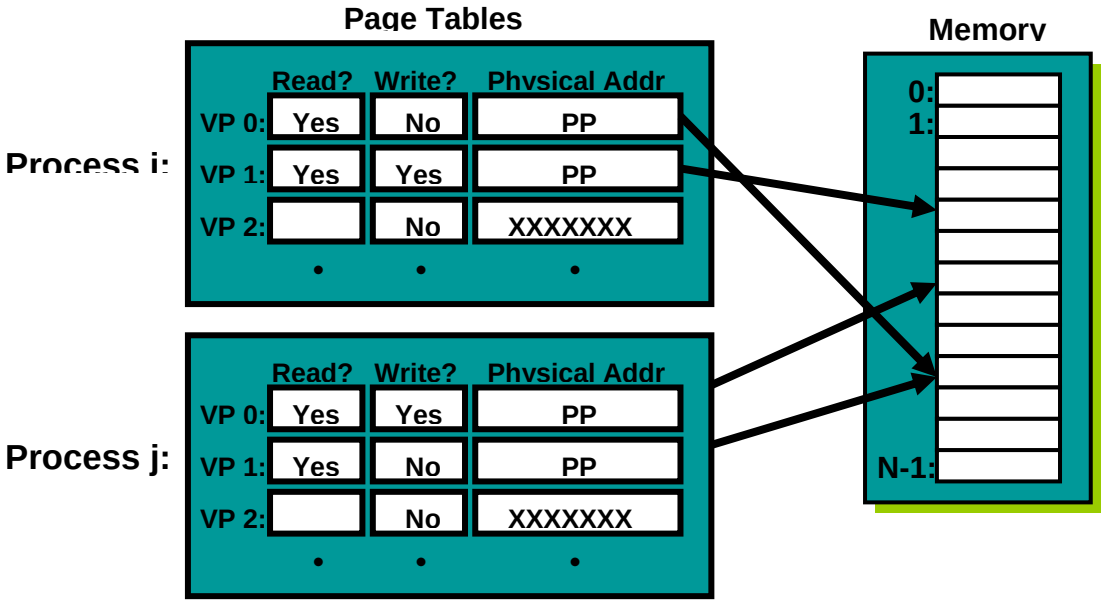

图6 页面属性示例

2.2.1 PAE（Physical Address Extensions）

在x86 CPU中，只有32位地址总线，也就意味着只有4G地址空间。为了实现在32位系统中使用更多的物理内存，Intel CPU提供了PAE（Physical Address Extensions）机制，这样可以使用超过4G物理内存。

PAE机制的打开，需要设置CR0、CR4控制寄存器和IA32_EFER MSR寄存器。设置值为CR0.PG = 1，CR4.PAE = 1和IA32_EFER.LME = 0。但PAE机制打开后，MMU会将32位线性地址转换为52位物理地址。尽管物理地址是52位（4PB），但线性地址仍然为32位，即进程可使用的物理内存不超过4GB。

2.2.2 PSE（Page Size Extensions）

前面提到x86/x86_64 CPU中提供的是段页式机制，也介绍了页面。页面大小虽然是固定的，但x86/x86_64 CPU支持两种页面大小4KB和4MB。大多数情况下OS内核使用的是4KB页面，但系统中多数进程使用内存较大时，如每次申请至少4MB，则使用4MB页面较为合适。

当控制寄存器CR4.PSE = 1时，页面大小是4MB。

2.3 TLB（Translation Lookaside Buffers）

在《Linux内存地址映射》中，详细介绍了CPU如何将逻辑地址转换为物理地址，这个过程需要多次访问物理内存，从内存中读取页面目录表、页面表。若有4级映射，从内存读取地址映射表的次数更多。这个过程相对CPU运算能力而言，是非常耗时的。

CPU可以将线性地址映射信息保存在TLB中，即TLB保存着线性地址与物理页面的对应关系，这样CPU下次访问该线性地址时，直接可以从TLB中找到对应的物理页面，不再需要线性地址到物理地址的繁琐映射过程，大大加快找到物理页面的速度。

我们对CPU的Cache非常熟悉，一般用缓存内存数据的指令。TLB也是一种Cache，也是在集成在CPU内部，只是它保存地址映射关系，不是数据或指令。

线性地址映射时，每次都首先会从TLB中查找，若命中，则直接读出物理页面地址；若未命中，再进行页式地址映射。

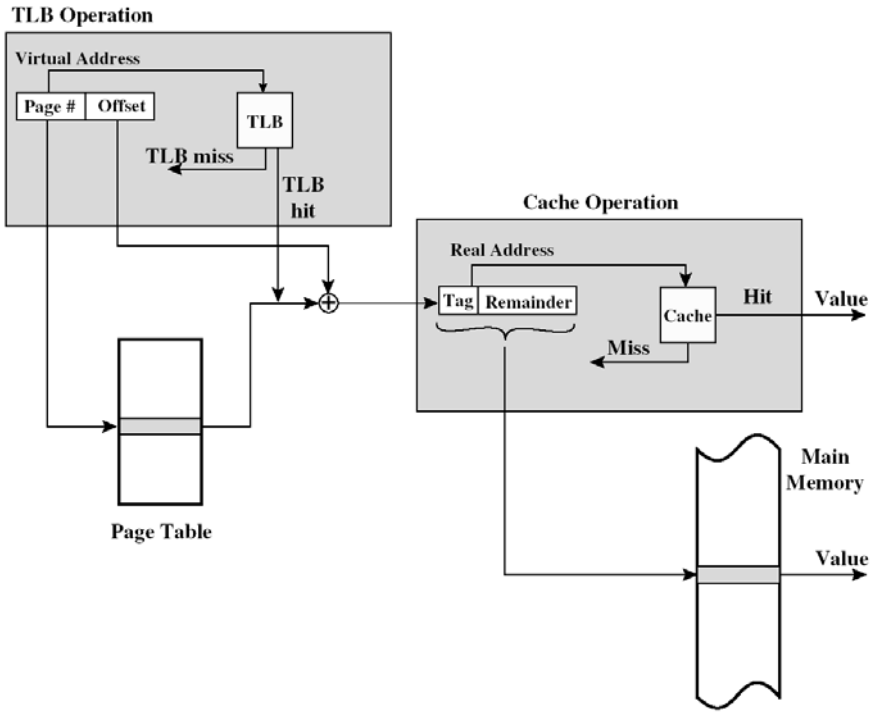

图7 Translation Lookaside Buffer和Cache

3 Linux内核对物理内存描述

Linux将物理内存按固定大小的页面（一般为4K）划分内存，在内核初始化时，会建立一个**全局struct page结构数组mem_map[ ]**。如系统中有76G物理内存，则物理内存页面数为76*1024*1024k/4K= 19922944个页面，mem_map[ ]数组大小19922944，即为数组中

每个元素和物理内存页面一一对应，整个数组就代表着系统中的全部物理页面。

在服务器中，存在NUMA架构（如Nehalem、Romly等），Linux将NUMA中内存访问速度一致（如按照内存通道划分）的部分称为一个**节点（Node）**，用struct pglist_data数据结构表示，通常使用时用它的typedef定义pg_data_t。系统中的每个结点都通过pgdat_list链表pg_data_t->node_next连接起来，该链接以NULL为结束标志。

每个结点又进一步分为许多块，称为**区域（zones）**。区域表示内存中的一块范围。区域用struct zone_struct数据结构表示，它的typedef定义为zone_t。

每个区域（Zone）中有多个**页面（Pages）**组成。节点、区域、页面三者关系如下图。

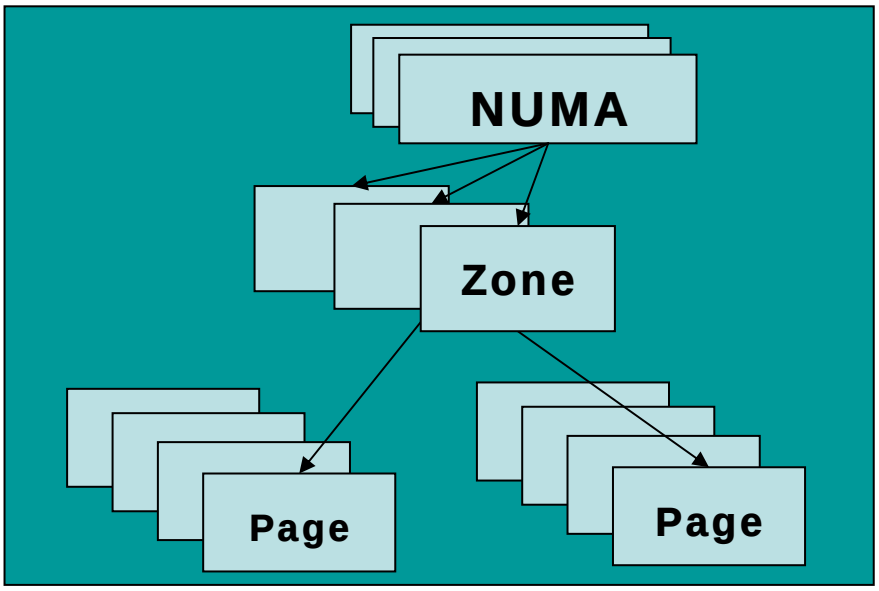

图8 节点、区域及页面关系图

3.1 节点（Node）

节点（Node），在linux中用struct pglist_data数据结构表示，通常使用时用它的typedef定义pg_data_t，数据结构定义在文件include/linux/mmzone.h中。

```
00630: /*
00631:  * The pg_data_t structure is used in machines with
CONFIG_DISCONTIGMEM
00632:  * (mostly NUMA machines?) to denote a higher-level memory zone
than the
00633:  * zone denotes.
00634:
00635:  * On NUMA machines, each NUMA node would have a pg_data_t to
describe
00636:  * it's memory layout.
00637:  *
```

```
00638:  * Memory statistics and page replacement data structures are
maintained on a
00639:  * per-zone basis.
00640:  */
00641: struct bootmem_data;
00642: typedef struct pglist_data {
00643:         struct zone node_zones[MAX_NR_ZONES];
00644:         struct zonelist node_zonelists[MAX_ZONELISTS];
00645:         int nr_zones;
00646: #ifdef CONFIG_FLAT_NODE_MEM_MAP       /* means !SPARSEMEM
*/
00647:         struct page *node_mem_map;
00648: #ifdef CONFIG_CGROUP_MEM_RES_CTLR
00649:         struct page_cgroup *node_page_cgroup;
00650: #endif
00651: #endif
00652:         struct bootmem_data *bdata;
00653: #ifdef CONFIG_MEMORY_HOTPLUG
00654:         /*
00655:          * Must be held any time you expect node_start_pfn,
node_present_pages
00656:          * or node_spanned_pages stay constant.  Holding this will also
00657:          * guarantee that any pfn_valid() stays that way.
00658:          *
00659:          * Nests above zone->lock and zone->size_seqlock.
00660:          */
00661:         spinlock_t node_size_lock;
00662: #endif
00663:         unsigned long node_start_pfn;
00664:         unsigned long node_present_pages; /* total number of physical
pages */
00665:         unsigned long node_spanned_pages; /* total size of physical page
00666:                                              range, including holes */
00667:         int node_id;
00668:         wait_queue_head_t kswapd_wait;
00669:         struct task_struct *kswapd;
00670:         int kswapd_max_order;
00671: } ?    end pglist_data ?  pg_data_t;
00672:
```

其中的成员变量含义如下：

node_zones：这个结点的区域有，在x86中有，ZONE_HIGHMEM，ZONE_NORMAL，ZONE_DMA；x86_64 CPU中区域有DMA、DMA32和NORMAL三部分。

node_zonelists：分配区域时的顺序，由函数free_area_init_core()调用mm/page_alloc.c中函数build_zonelists()设置；

nr_zones：区域的数量，值的范围为1～3；并不是所有的结点都有三个区域；

node_mem_map：该结点的第一页面在全局变量mem_map数组中地址；

bdata：仅在系统启动分配内存时使用；

node_start_pfn：该结点的起始物理页面号；

node_present_pages：该结点中的总共页面数；

node_spanned_pages：该节点中所有物理页面数，包括内存空洞（如部分地址为外设I/O使用）。

node_id：结点ID，从0开始；

当分配一个页面时，linux使用本地结点分配策略，从运行的CPU最近的一个结点分配。因为进程倾向于在同一个CPU上运行，使用内存时也就更可能使用本结点的空间。

对于象PC之类的UMA系统，仅有一个静态的pg_data_t结构，变量名为contig_page_data。

```
04850: #ifndef CONFIG_NEED_MULTIPLE_NODES
04851: struct pglist_data __refdata contig_page_data = { .bdata =
&bootmem_node_data[0] };
04852: EXPORT_SYMBOL(contig_page_data);
04853: #endif
```

遍历所有节点可以使用for_each_online_pgdat(pgdat)来实现。

```
00830: /**
00831:  * for_each_online_pgdat - helper macro to iterate over all online
nodes
00832:  * @pgdat - pointer to a pg_data_t variable
00833:  */
00834: #define for_each_online_pgdat(pgdat)                    \
00835:         for (pgdat = first_online_pgdat();              \
00836:              pgdat;                                     \
00837:              pgdat = next_online_pgdat(pgdat))
```

3.2 区域（Zone）

3.2.1 区域类型

节点（Node）下面可以有多个区域， 共有以下几种类型：

```
00198: enum zone_type {
00199: #ifdef CONFIG_ZONE_DMA
00218:         ZONE_DMA,
00219: #endif
00220: #ifdef CONFIG_ZONE_DMA32
00221:         /*
00222:          * x86_64 needs two ZONE_DMAs because it supports devices
```

```
 that are
00223:          * only able to do DMA to the lower 16M but also 32 bit devices
that
00224:          * can only do DMA areas below 4G.
00225:          */
00226:         ZONE_DMA32,
00227: #endif
00233:         ZONE_NORMAL,
00234: #ifdef CONFIG_HIGHMEM
00235:         /*
00236:          * A memory area that is only addressable by the kernel through
00237:          * mapping portions into its own address space. This is for
example
00238:          * used by i386 to allow the kernel to address the memory beyond
00239:          * 900MB. The kernel will set up special mappings (page
00240:          * table entries on i386) for each page that the kernel needs to
00241:          * access.
00242:          */
00243:         ZONE_HIGHMEM,
00244: #endif
00245:         ZONE_MOVABLE,
00246:         __MAX_NR_ZONES
00247: };
```

ZONE_DMA

是低内存的一块区域，这块区域由标准工业架构（Industry Standard Architecture）设备使用，适合DMA内存。这部分区域大小和CPU架构有关，在x86架构中，该部分区域大小限制为16MB。

ZONE_DMA32

该部分区域为适合支持32位地址总线的DMA内存空间。很显然，该部分仅在64位系统有效，在32位系统中，这部分区域为空。在x86-64架构中，这部分的区域范围为0～4GB。

ZONE_NORMAL

属于ZONE_NORMAL的内存被内核直接映射到线性地址。这部分区域仅表示可能存在这部分区域，如在64位系统中，若系统只有4GB物理内存，则所有的物理内存都属于ZONE_DMA32，而ZONE_NORMAL区域为空。

许多内核操作都仅在ZONE_NORMAL内存区域进行，所以这部分是系统性能关键的地方。

ZONE_HIGHMEM

是系统中剩下的可用内存，但因为内核的地址空间有限，这部分内存不直接映射到内核。详细内容可参考文档《Linux内核高端内存》（http://ilinuxkernel.com/?p=1013）。

在x86架构中内存有三种区域：ZONE_DMA，ONE_NORMAL，ZONE_HIGHMEM。不同类型的区域适合不同需要。在32位系统中结构中，1G（内核空间）/3G（用户空间）地址空间划分时，三种类型的区域如下：

- ZONE_DMA　　内存开始的16MB
- ZONE_NORMAL　16MB~896MB
- ZONE_HIGHMEM　896MB ~ 结束

4G（内核空间）/4G（用户空间）地址空间划分时，三种类型区域划分为：

- ZONE_DMA　　内存开始的16MB
- ZONE_NORMAL　16MB~3968MB
- ZONE_HIGHMEM　3968MB ~ 结束

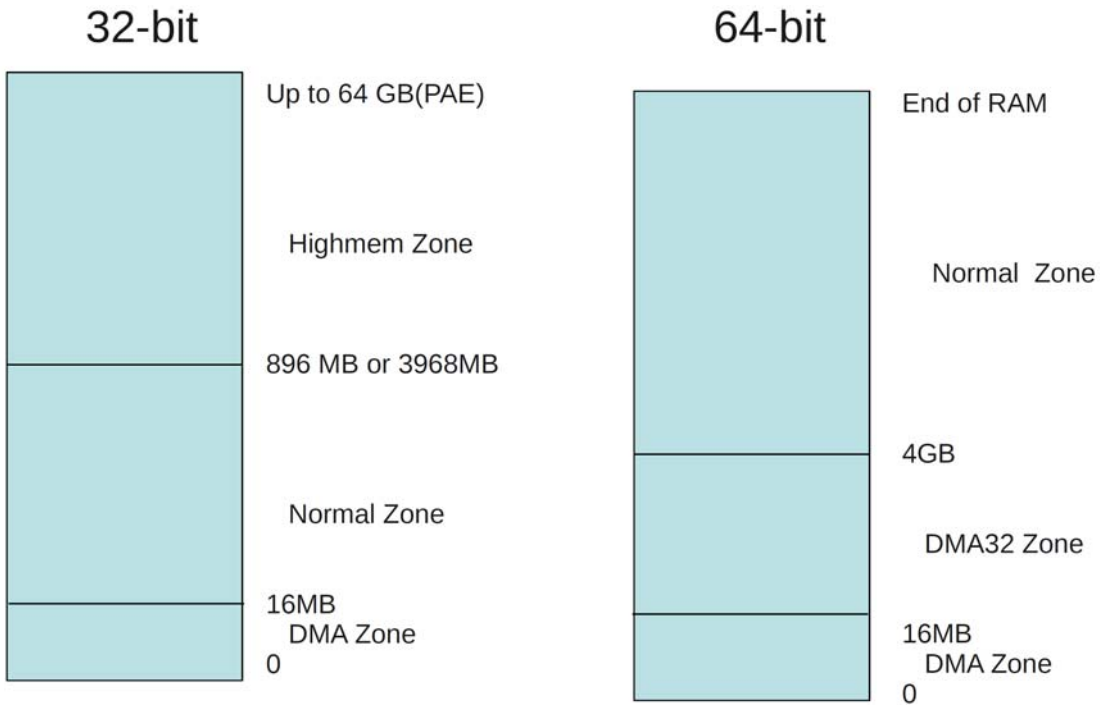

图9 32位和64位系统中内存区域划分

在64位Linux系统中，内存只有三个区域DMA、DMA32和NORMAL

- ZONE_DMA　　内存开始的16MB

- ZONE_DMA32　　16MB～4GB
- ZONE_NORMAL　4GB ~ 结束

对于64位系统中，为什么不存在高端内存（ZONE_HIGHMEM），可参考文档《Linux内核高端内存》（http://ilinuxkernel.com/?p=1013）。

3.2.2　struct zone结构体

每块区域的描述结构为struct zone。zone跟踪记录一些信息，如页面的使用统计、空闲页面及锁等，数据结构定义在文件include/linux/mmzone.h中。

```
00287: struct zone {
00288:      /* Fields commonly accessed by the page allocator */
00289:
00290:      /* zone watermarks, access with *_wmark_pages(zone) macros
*/
00291:      unsigned long watermark[NR_WMARK];
 ...
00310:      unsigned long       lowmem_reserve[MAX_NR_ZONES];
00311:
00312: #ifdef CONFIG_NUMA
00313:      int node;
00314:      /*
00315:       * zone reclaim becomes active if more unmapped pages exist.
00316:       */
00317:      unsigned long       min_unmapped_pages;
00318:      unsigned long       min_slab_pages;
00319:      struct per_cpu_pageset    *pageset[NR_CPUS];
00320: #else
00321:      struct per_cpu_pageset    pageset[NR_CPUS];
00322: #endif
00323:      /*
00324:       * free areas of different sizes
00325:       */
00326:      spinlock_t          lock;
00327: #ifdef CONFIG_MEMORY_HOTPLUG
00328:      /* see spanned/present_pages for more description */
00329:      seqlock_t       span_seqlock;
00330: #endif
00331:      struct free_area     free_area[MAX_ORDER];
  ... ...
00353:      /* Fields commonly accessed by the page reclaim scanner */
00354:      spinlock_t          lru_lock;
00355:      struct zone_lru {
00356:           struct list_head list;
```

```
00357:      } lru[NR_LRU_LISTS];
00358:
00359:      struct zone_reclaim_stat reclaim_stat;
00360:
00361:      unsigned long       pages_scanned;      /* since last reclaim */
00362:      unsigned long       flags;              /* zone flags, see below */
00363:
00364:      /* Zone statistics */
00365:      atomic_long_t       vm_stat[NR_VM_ZONE_STAT_ITEMS];
00366:
00380:      int prev_priority;
00386:      unsigned int inactive_ratio;
00416:      wait_queue_head_t * wait_table;
00417:      unsigned long       wait_table_hash_nr_entries;
00418:      unsigned long       wait_table_bits;
00419:
00420:      /*
00421:       * Discontig memory support fields.
00422:       */
00423:      struct pglist_data  *zone_pgdat;
00424:      /* zone_start_pfn == zone_start_paddr >> PAGE_SHIFT */
00425:      unsigned long       zone_start_pfn;
00437:      unsigned long       spanned_pages;    /* total size, including holes
*/
00438:      unsigned long       present_pages;    /* amount of memory
(excluding holes) */
00439:
00440:      /*
00441:       * rarely used fields:
00442:       */
00443:      const char          *name;
00444:
00445:      unsigned long padding[16];
00446: } ?   end zone ? ____cacheline_internodealigned_in_smp;
```

结构体中主要的成员变量含义：

lock：spinlock防止对区域的并发访问；

pages_min，***pages_low***和***pages_high***：区域的“水准”，即区域页面分配的不同水准；

lowmem_reserve：针对每个区域保存的物理页面数量，保证在任何条件下，申请内存都不会失败；

free_area：buddy分配器使用的空闲区域位图；

wait_table, wait_table_bits和wait_table_hash_nr_entries：等待空闲页面的进程hash表，这个对wait_on_page()和unlock_page()重要；队列上的进程等待某些条件，当条件满足时，内核就会通知进程，进程就可以继续执行。

zone_pdgat：指向父亲pg_data_t；

zone_start_pfn：该区域结点的物理页面号；

zone_present_pages：该结点中的总共页面数；

zone_spanned_pages：该节点中所有物理页面数，包括内存空洞（如部分地址为外设I/O使用）。

name：区域的名字，“DMA“，“DMA32”，”Normal“，“HighMem”；

3.3 页面（Page）

系统内存由固定的块组成，称为页帧，每个页帧由struct page结构描述。内核在初始化时，会根据内存的大小计算出由多少页帧，每个页帧都会有一个page结构与之对应，这些信息保存在全局数组变量mem_map中。mem_map通常存储在ZONE_NORMAL区域中，在内存较小的机器中，会保存在加载内核镜像后的一片保留空间里。有多少个物理页面，就会有多个struct page结构，如系统安装128GB物理内存，struct page结构体大小为40字节，则mem_map[ ]数组就占用物理内存大小为128*1024*1024k/4k * 40 = 1280MB，即Linux内核要使用1280MB物理内存来保存mem_map[ ]数组，这部分内存是不可被使用的。因此struct page结构体大小不能设计很大。

3.3.1 struct page结构体

struct page数据结构定义在文件include/linux/mm_types.h文件中。

```
00040: struct page {
00041:      unsigned long flags;          /* Atomic flags, some possibly
00042:                                    * updated asynchronously */
00043:      atomic_t _count;              /* Usage count, see below. */
00044:      union {
00045:          atomic_t _mapcount;       /* Count of ptes mapped in mms,
00046:                                    * to show when page is mapped
00047:                                    * & limit reverse map searches.
00048:                                    */
00049:          struct {        / * SLUB */
00050:              u16 inuse;
00051:              u16 objects;
00052:          };
00053:      };
00054:      union {
00055:        struct {
00056:          unsigned long private;
00063:           struct address_space *mapping;
00070:        };
00071: #if USE_SPLIT_PTLOCKS
```

```
00072:          spinlock_t ptl;
00073: #endif
00074:          struct kmem_cache *slab; /* SLUB: Pointer to slab */
00075:          struct page *first_page;    /* Compound tail pages */
00076:      } ?    end {anon_union} ?    ;
00077:      union {
00078:          pgoff_t index;      /* Our offset within mapping. */
00079:          void *freelist;     /* SLUB: freelist req. slab lock */
00080:      };
00081:      struct list_head lru;
00094: #if defined(WANT_PAGE_VIRTUAL)
00095:      void *virtual;          /* Kernel virtual address (NULL if
00096:                                 not kmapped, ie. highmem) */
00097: #endif /* WANT_PAGE_VIRTUAL */
00098: #ifdef CONFIG_WANT_PAGE_DEBUG_FLAGS
00099:      unsigned long debug_flags;    /* Use atomic bitops on this */
00100: #endif
00101:
00102: #ifdef CONFIG_KMEMCHECK
00107:      void *shadow;
00108: #endif
00109: } ?    end page ?    ;
00110:
```

page结构中的参数含义如下：

flags：描述页面状态的标志。所有的标志在include/linux/page-flags.h中定义，主要标志包括PG_locked、PG_error、PG_referenced、PG_uptodate、PG_active、PG_dirty、PG_lru等。系统中定义了许多宏来测试、清除、设置标志中的不同bit。

mapping：当文件或者设备映射到内存中时，它们的inode结构就会和address_space相关联。当页面属于一个文件时，mapping就会指向这个地址空间。如果这个页面是匿名的且映射开启，则address_space就是swapper_space，swapper_space是管理交换地址空间的。

index：这个值有两个用处，具体用处取决于页面的状态。若页面是属于一个文件的映射，则index是该页面在文件中的偏移量。若页面是交换缓冲区的一部分，则index是页面在address_space交换地址空间的偏移量（swapper_space）。另外，当一些页面被一个进程回收时，该回收区域的级别（2的幂次方数量回收的页面）保存在index中，这个值在函数__free_pages_ok()中设置。

_count：引用该页面的计数。当该值达到0时，页面可以被回收。当大于0时，意味着有一个或多个进程正在使用该页面。

_mapcount：页面表总共有多少项指向该页面。。

lru：为页面替换策略，可以被换出的页面可能存在于active_list或者inactive_list（在page_alloc.c中定义）。这是LRU（Least Recently Used）链表头。

virtual：正常情况下只有处于ZONE_NORMAL的页面被内核直接映射。对于ZONE_HIGHMEM区域的页面，内核使用kmap()来映射页面。当页面被映射后，virtual时它的虚地址。

3.3.2 页面标志

页面标志尤为重要，则内存分配与回收、I/O操作等重要内核活动过程中都会使用到页面标志。所有的标志在include/linux/page-flags.h中定义。

```
00075: enum pageflags {
00076:         PG_locked,          /* Page is locked. Don't touch. */
00077:         PG_error,
00078:         PG_referenced,
00079:         PG_uptodate,
00080:         PG_dirty,
00081:         PG_lru,
00082:         PG_active,
00083:         PG_slab,
00084:         PG_owner_priv_1,    /* Owner use. If pagecache, fs may use*/
00085:         PG_arch_1,
00086:         PG_reserved,
00087:         PG_private,         /* If pagecache, has fs-private data */
00088:         PG_private_2,           /* If pagecache, has fs aux data */
00089:         PG_writeback,           /* Page is under writeback */
00090: #ifdef CONFIG_PAGEFLAGS_EXTENDED
00091:         PG_head,            /* A head page */
00092:         PG_tail,        /* A tail page */
00093: #else
00094:         PG_compound,            /* A compound page */
00095: #endif
00096:         PG_swapcache,           /* Swap page: swp_entry_t in private */
00097:         PG_mappedtodisk,        /* Has blocks allocated on-disk */
00098:         PG_reclaim,         /* To be reclaimed asap */
00099:         PG_buddy,           /* Page is free, on buddy lists */
00100:         PG_swapbacked,          /* Page is backed by RAM/swap */
00101:         PG_unevictable,         /* Page is "unevictable"  */
00102: #ifdef CONFIG_MMU
00103:         PG_mlocked,         /* Page is vma mlocked */
00104: #endif
00105: #ifdef CONFIG_ARCH_USES_PG_UNCACHED
00106:         PG_uncached,            /* Page has been mapped as uncached */
00107: #endif
00108: #ifdef CONFIG_MEMORY_FAILURE
00109:         PG_hwpoison,            /* hardware poisoned page. Don't touch */
00110: #endif
```

```
00111: #ifdef CONFIG_TRANSPARENT_HUGEPAGE
00112:         PG_compound_lock,
00113: #endif
00114:         __NR_PAGEFLAGS,
00115:
00116:         /* Filesystems */
00117:         PG_checked = PG_owner_priv_1,
00123:         PG_fscache = PG_private_2, /* page backed by cache */
00124:
00125:         /* XEN */
00126:         PG_pinned = PG_owner_priv_1,
00127:         PG_savepinned = PG_dirty,
00128:
00129:         /* SLOB */
00130:         PG_slob_free = PG_private,
00131:
00132:         /* SLUB */
00133:         PG_slub_frozen = PG_active,
00134:         PG_slub_debug = PG_error,
00135: };
```

这里解释一下重要的几个页面标志：

PG_locked：页面是否被锁住，若该位设置了该位，则不允许内核其他部分访问该页面。这用来防止内存管理过程中遇到的竞争条件，如当从硬盘读取数据到一个页面时，就不允许其他内核部分访问该页面，因为读数据的过程中，其他内核部分能访问的话，则读取到的数据是不完整的。

PG_error：I/O出错，且操作和页面有关，就设置该标志。

PG_referenced和PG_active：控制系统使用页面的活跃程度。这个信息对swap系统选择待交换出的页面非常重要。

PG_update：表示成功完成从块设备上读取一个页面的数据。该标志和块设备I/O操作有关。

PG_dirty：当内存页面中的数据和块设备上的数据不一致时，就设置该标志。在写数据到块设备时，为了提高将来的读性能，数据并不是立即回写到块设备上，而只是设置页面脏标志，表示该页面数据需要回写。

PG_lru：该标志用来实现页面回收和交换。

PG_highmem：表示该页面为属于高端内存。

4 Linux物理内存描述信息查看

4.1 /proc/zoneinfo

```
root@yiquan-ThinkPad-X200:/proc# cat zoneinfo
Node 0, zone      DMA
  pages free     3977
        min      66
        low      82
        high     99
        scanned  0
        spanned  4080
        present  3913
    nr_free_pages 3977
    nr_inactive_anon 0
    nr_active_anon 0
    nr_inactive_file 0
    nr_active_file 0
    nr_unevictable 0
    nr_mlock     0
    nr_anon_pages 0
    nr_mapped    0
    nr_file_pages 0
    nr_dirty     0
    nr_writeback 0
    nr_slab_reclaimable 0
    nr_slab_unreclaimable 0
    nr_page_table_pages 0
    nr_kernel_stack 0
    nr_unstable  0
    nr_bounce    0
```

4.2 /proc/pagetypeinfo

```
root@yiquan-ThinkPad-X200:/proc# cat pagetypeinfo
Page block order: 9
Pages per block:  512

Free pages count per migrate type at order       0      1      2      3      4      5      6      7      8      9     10
Node    0, zone      DMA, type    Unmovable      1      0      0      1      2      1      1      0      1      0      0
Node    0, zone      DMA, type  Reclaimable      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone      DMA, type      Movable      0      0      0      0      0      0      0      0      0      0      3
Node    0, zone      DMA, type      Reserve      0      0      0      0      0      0      0      0      0      1      0
Node    0, zone      DMA, type      Isolate      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone    DMA32, type    Unmovable     30     66     29      2      2      1      0      1      1      1      0
Node    0, zone    DMA32, type  Reclaimable     62     17     67     36     36     14     10      4      3      0      0
Node    0, zone    DMA32, type      Movable      8      7      2      2      2      0      2      1      1      1    715
Node    0, zone    DMA32, type      Reserve      0      0      0      0      0      0      0      0      0      0      1
Node    0, zone    DMA32, type      Isolate      0      0      0      0      0      0      0      0      0      0      0
Node    0, zone   Normal, type    Unmovable     21     62     53      6      2      2      1      1      1      0      0
Node    0, zone   Normal, type  Reclaimable     70     16     41     13      7      3      2      0      1      1      0
Node    0, zone   Normal, type      Movable      1      1      0      1      1      1      0      0      1      1      1
Node    0, zone   Normal, type      Reserve      0      0      0      0      0      0      0      0      0      0      1
Node    0, zone   Normal, type      Isolate      0      0      0      0      0      0      0      0      0      0      0

Number of blocks type     Unmovable  Reclaimable      Movable      Reserve      Isolate
Node 0, zone      DMA            1            0            6            1            0
Node 0, zone    DMA32           10           12         1504            2            0
Node 0, zone   Normal           56           82          340            2            0
root@yiquan-ThinkPad-X200:/proc#
```

4.3 /sys/devices/system/node/node*/meminfo

```
root@yiquan-ThinkPad-X200:/sys/devices/system/node/node0# cat meminfo
Node 0 MemTotal:        4087776 kB
Node 0 MemFree:         2878244 kB
Node 0 MemUsed:         1209532 kB
Node 0 Active:           387284 kB
Node 0 Inactive:         535136 kB
Node 0 Active(anon):     322828 kB
Node 0 Inactive(anon):   178240 kB
Node 0 Active(file):      64456 kB
Node 0 Inactive(file):   356896 kB
Node 0 Unevictable:           0 kB
Node 0 Mlocked:               0 kB
Node 0 Dirty:                32 kB
Node 0 Writeback:             0 kB
Node 0 FilePages:        600164 kB
Node 0 Mapped:            83092 kB
Node 0 AnonPages:        322412 kB
Node 0 Shmem:            178816 kB
Node 0 KernelStack:        2560 kB
Node 0 PageTables:        22560 kB
Node 0 NFS_Unstable:          0 kB
Node 0 Bounce:                0 kB
Node 0 WritebackTmp:          0 kB
Node 0 Slab:              49648 kB
Node 0 SReclaimable:      24700 kB
Node 0 SUnreclaim:        24948 kB
Node 0 AnonHugePages:         0 kB
```

4.4 echo m > /proc/sysrq-trigger

[root@RH2285 sys]# echo m > /proc/sysrq-trigger

[root@RH2285 sys]#

SysRq : Show Memory

Mem-Info:

Node 0 DMA per-cpu:

... ...

active_anon:9285 inactive_anon:2 isolated_anon:0

active_file:9114 inactive_file:29422 isolated_file:0

unevictable:0 dirty:0 writeback:0 unstable:0

free:18271427 slab_reclaimable:4624 slab_unreclaimable:20536

mapped:5436 shmem:59 pagetables:1370 bounce:0

Node 0 DMA free:15532kB min:16kB low:20kB high:24kB active_anon:0kB inactive_anon:0kB active_file:0kB inactive_file:0kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:15120kB mlocked:0kB dirty:0kB writeback:0kB mapped:0kB shmem:0kB slab_reclaimable:0kB slab_unreclaimable:0kB kernel_stack:0kB pagetables:0kB unstable:0kB bounce:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no

lowmem_reserve[]: 0 2991 36321 36321

Node 0 DMA32 free:2387360kB min:3708kB low:4632kB high:5560kB active_anon:0kB inactive_anon:0kB active_file:0kB inactive_file:0kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:3063584kB mlocked:0kB dirty:0kB writeback:0kB mapped:0kB shmem:0kB slab_reclaimable:0kB slab_unreclaimable:0kB kernel_stack:0kB pagetables:0kB unstable:0kB bounce:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no

lowmem_reserve[]: 0 0 33330 33330

Node 0 Normal free:33934856kB min:41312kB low:51640kB high:61968kB active_anon:25156kB inactive_anon:0kB active_file:29252kB inactive_file:93552kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:34129920kB mlocked:0kB dirty:0kB writeback:0kB mapped:17032kB shmem:112kB

slab_reclaimable:11712kB slab_unreclaimable:61124kB kernel_stack:3168kB pagetables:2744kB unstable:0kB bounce:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no
lowmem_reserve[]: 0 0 0 0
Node 1 Normal free:36747960kB min:45068kB low:56332kB high:67600kB active_anon:11984kB inactive_anon:8kB active_file:7204kB inactive_file:24136kB unevictable:0kB isolated(anon):0kB isolated(file):0kB present:37232640kB mlocked:0kB dirty:0kB writeback:0kB mapped:4712kB shmem:124kB slab_reclaimable:6784kB slab_unreclaimable:21020kB kernel_stack:280kB pagetables:2736kB unstable:0kB bounce:0kB writeback_tmp:0kB pages_scanned:0 all_unreclaimable? no
lowmem_reserve[]: 0 0 0 0
*Node 0 DMA: 1*4kB 1*8kB 0*16kB 1*32kB 2*64kB 0*128kB 0*256kB 0*512kB 1*1024kB 1*2048kB 3*4096kB = 15532kB*
*Node 0 DMA32: 8*4kB 16*8kB 4*16kB 8*32kB 11*64kB 6*128kB 4*256kB 9*512kB 8*1024kB 6*2048kB 576*4096kB = 2387360kB*
*Node 0 Normal: 276*4kB 413*8kB 193*16kB 54*32kB 34*64kB 15*128kB 10*256kB 6*512kB 5*1024kB 2*2048kB 8278*4096kB = 33934856kB*
*Node 1 Normal: 495*4kB 692*8kB 228*16kB 87*32kB 39*64kB 18*128kB 19*256kB 11*512kB 4*1024kB 3*2048kB 8962*4096kB = 36747836kB*
38601 total pagecache pages
0 pages in swap cache
Swap cache stats: add 0, delete 0, find 0/0
Free swap = 0kB
Total swap = 0kB
18874352 pages RAM
314941 pages reserved
27283 pages shared
262676 pages non-shared
[root@RH2285 sys]#