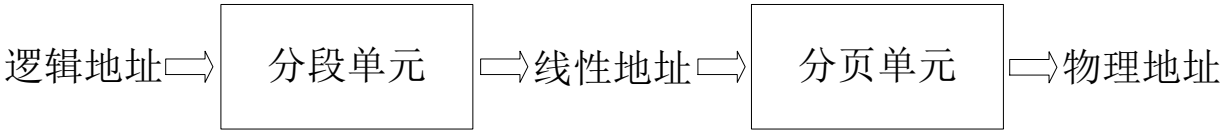
1. 内存地址

1.1. 内存地址概要

1) 逻辑地址：分段，如代码段、数据段等。通过段地址+偏移量寻址。
2) 线性地址(虚拟地址)：分页。线性地址的连续，不代表对应的物理地址也是连续的。
3) 物理地址：申请和释放的物理地址都是连续的。

内存控制单元（MMU）通过分段单元（segmentation unit）的硬件电路将逻辑地址转换成线性地址；再通过分页单元（paging unit）的硬件电路把线性地址转换成物理地址。如下图：

图 1-1 内存地址转换



1.2. TLB

TLB(Translation Lookaside Buffer)是CPU中的一个硬件单元，用来缓存部分页表，以加快虚拟地址到物理地址的转换速度。

图 1-2 TLB、页表、内存的关系

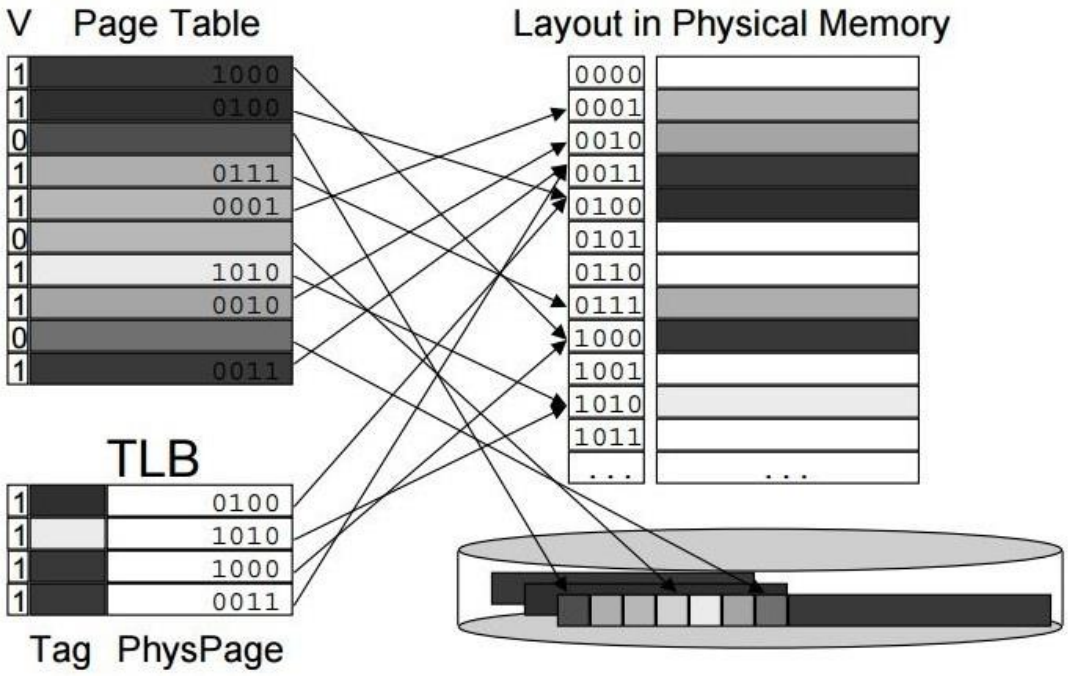
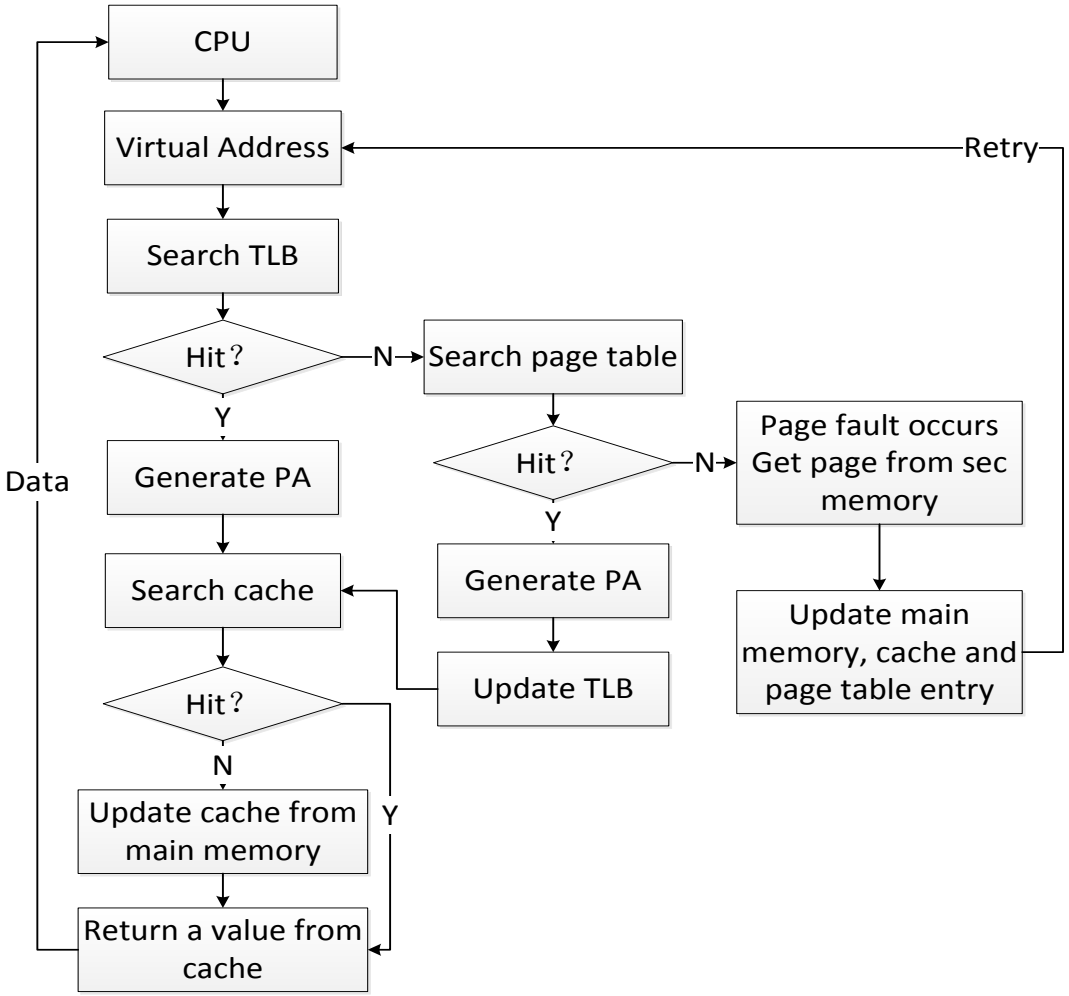


図 1-3 虚拟地址到物理地址的转换过程



可以通过perf工具查看TLB的命中情况:

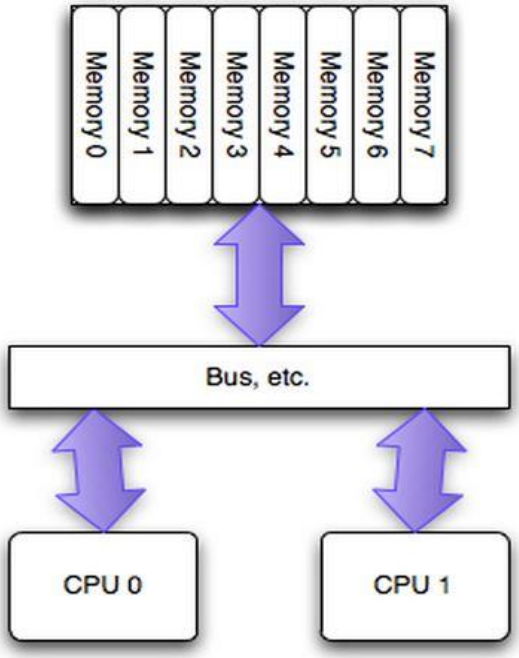
```
# perf list | grep TLB
  dTLB-loads                                         [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-stores                                        [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  iTLB-loads                                         [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
# perf stat -e xxx -p xxxx
```

2. NUMA

2.1. NUMA概要

NUMA（Non Uniform Memory Access），即非一致内存访问，是针对UMA（Uniform Memory Access）提出的。

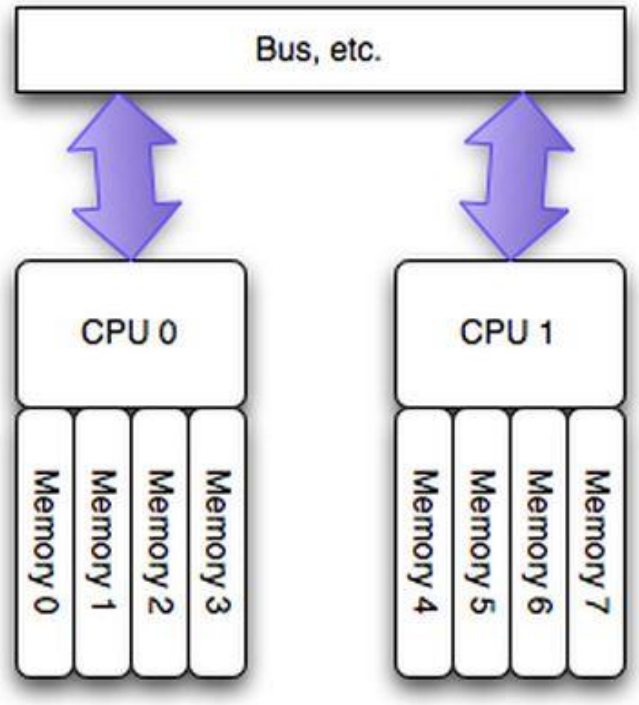
图 2-4 UMA架构



UMA架构下：

1） 多个CPU和主存（Memory）通过一条系统总线通信。
2） 随着系统规模的扩大，CPU数量不断增加，对系统总线的争抢现象越发严重，系统总线带宽成为了多CPU访问内存的瓶颈。

图 2-5 NUMA架构



NUMA架构下：

1) 一个NUMA系统由多个NUMA Node组成，其中每个Node可以拥有多个CPU，但是只有一个内存控制器。
2) 每个CPU都有自己的主存，CPU与自己的主存在物理上距离较近，访问速度较快，而访问远端内存的速度相对较慢。

2.2. numastat

通过numastat命令查看机器当前的NUMA相关信息：

```
# numastat -mn
Per-node system memory usage (in MBs):
                          Node 0          Node 1           Total
                 --------------- --------------- ---------------
MemTotal               16350.26        16384.00        32734.26
MemFree                 4011.92         8149.07        12160.99
MemUsed                12338.34         8234.93        20573.27
Active                 10940.15         5104.49        16044.64
Inactive                 625.55         2484.86         3110.40
Active(anon)            8467.86           60.22         8528.09
Inactive(anon)             0.12            8.09            8.22
Active(file)            2472.29         5044.27         7516.55
Inactive(file)           625.42         2476.76         3102.18
Unevictable                0.00            0.00            0.00
Mlocked                    0.00            0.00            0.00
Dirty                     13.81            0.70           14.51
Writeback                  0.00            0.00            0.00
FilePages               3098.45         7529.79        10628.24
```

```
Mapped                        31.95          34.68          66.63
AnonPages                   8467.53          59.30        8526.83
Shmem                         0.75           8.76           9.51
KernelStack                   8.16           1.86          10.02
PageTables                   23.99           4.69          28.68
NFS_Unstable                  0.00           0.00           0.00
Bounce                        0.00           0.00           0.00
WritebackTmp                  0.00           0.00           0.00
Slab                        103.62         254.90         358.52
SReclaimable                 59.85         226.38         286.23
SUnreclaim                   43.77          28.52          72.29
AnonHugePages              7526.00           6.00        7532.00
HugePages_Total               0.00           0.00           0.00
HugePages_Free                0.00           0.00           0.00
HugePages_Surp                0.00           0.00           0.00

Per-node numastat info (in MBs):
                          Node 0          Node 1           Total
                 --------------- --------------- ---------------
Numa_Hit                26730.49        47227.21        73957.70
Numa_Miss                   0.00          865.43          865.43
Numa_Foreign              865.43            0.00          865.43
Interleave_Hit             53.91           53.40          107.31
Local_Node              26701.71        47154.44        73856.15
Other_Node                 28.77          938.21          966.98
#
```

注：

1) Numa_Hist: numa_hit is memory successfully allocated on this node as intended.
2) Numa_Miss: numa_miss is memory allocated on this node despite the process preferring some different node. Each numa_miss has a numa_foreign on another node.
3) Numa_Foreign: numa_foreign is memory intended for this node, but actually allocated on some different node. Each numa_foreign has a numa_miss on another node.

2.3. numactl

通过numactl命令控制内存访问：

```
# numactl --membind 1 --cpunodebind 1 /usr/bin/mysqld_safe &
#
```

2.4. 效果确认

Benchmark: NUMA-STREAM-master.zip。本次检证启动2个线程，每个Node上运行1个线程。

1) local_access

```
Array size = 800000000
Total memory required = 18310.5 MB.
Each test is run 2 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Number of Threads requested = 2
Number of available nodes = 2
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       20088.3619       0.6372       0.6372       0.6372
Scale:      20334.7562       0.6295       0.6295       0.6295
Add:        22177.5087       0.8657       0.8657       0.8657
Triad:      22254.9834       0.8627       0.8627       0.8627
```

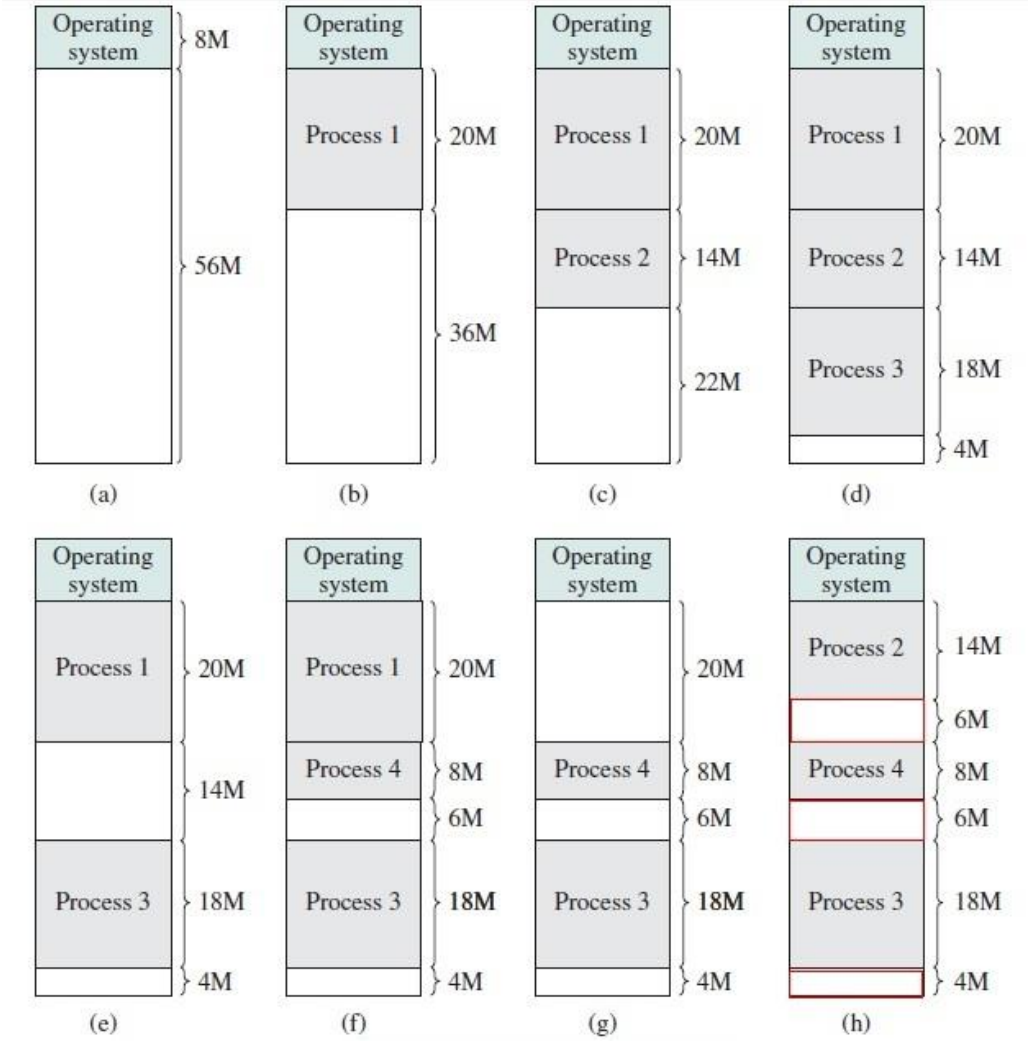
2) remote_access

```
Array size = 800000000
Total memory required = 18310.5 MB.
Each test is run 2 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Number of Threads requested = 2
Number of available nodes = 2
Execution will be non-NUMA aware.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       11569.7657       1.1063       1.1063       1.1063
Scale:      15252.5302       0.8392       0.8392       0.8392
Add:        16097.5506       1.1927       1.1927       1.1927
Triad:      15056.9698       1.2752       1.2752       1.2752
```

3. 内存外碎片

3.1. 外碎片概要

图 3-6 内存外碎片举例



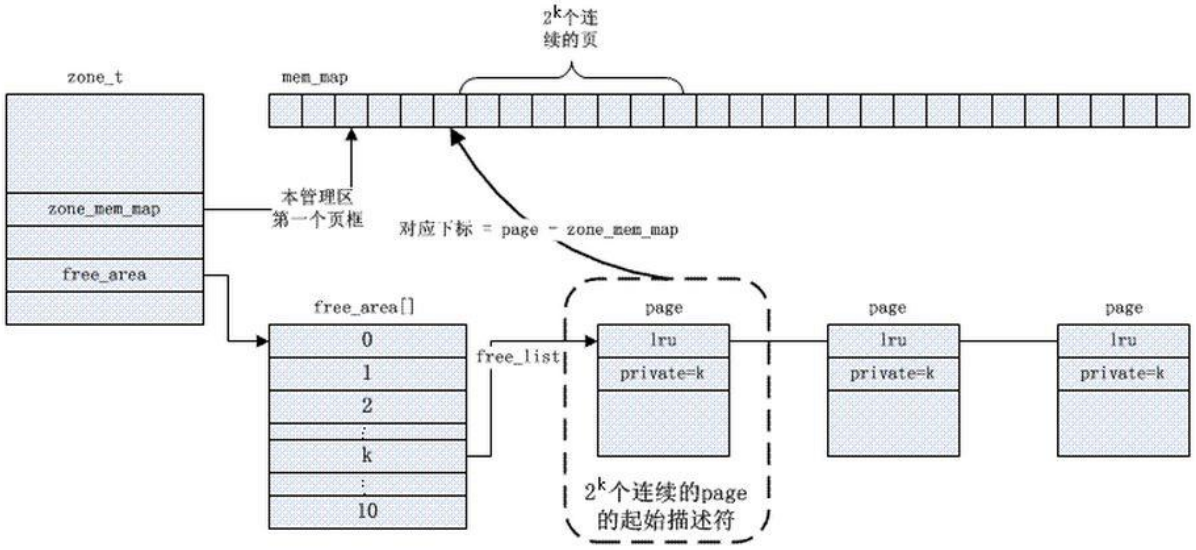
避免外碎片的方法有：

1) 非连续内存的分配。利用分页单元，将非连续的空闲页框映射到连续的线性地址空间。该方法需要频繁地刷新TLB和页表，效率低。
2) 用一种有效的方法来监视内存，保证在内核只要申请一小块内存的情况下，不会从大块的连续空闲内存中截取一段过来，从而保证了大块内存的连续性和完整性。Linux的伙伴系统(buddy system)采用该方法。

3.2. 伙伴算法

3.2.1. 伙伴算法概要

图 3-7 伙伴算法



说明：

1) 伙伴算法把所有的空闲页框分组为11个块链表。
2) 每个块链表分别包含大小为2^0, 2^1, 2^2, …, 2^10个连续页框的页框块，分别对应大小为4 KB, 8KB, 16KB, …, 4MB的连续物理内存。
3) 每个页框块的第一个页框的物理地址是该块大小的整数倍。

通过以下命令查看伙伴系统信息：

```
# cat /proc/buddyinfo
Node 0, zone      DMA      1      1      1      0      2      1      1      0      1      1      3
Node 0, zone    DMA32  13463  11871   9801   7443   5489   3670   2392   1198    362     38      0
Node 0, zone   Normal  23054  18629  15288  11305   8012   5222   3377   1492    408     52      3
#
```

3.2.2. 内存分配过程

假设请求分配4个页面，伙伴算法的页面分配过程如下：

1) 在第2(2^2=4)个组中寻找空闲块;
2) 如果第2个组中没有空闲块，就到第3(2^3=8)个组中寻找;
3) 假设第3个组中找到空闲块，就把其中的4个页面分配出去，剩余的4个页面放到第2个组中;
4) 如果第三个组还是没有空闲块，就到第4(2^4=16)个组中寻找。如果在第4个组找到空闲块，把其中的4个页面分配出去，剩余的12个页面被分成两部分，其中的8个页面放到第3个组，另外4个页面放到第2个组...依次类推;
5) 如果直到最后一个组(2^10=1024)也没有找到空闲块，则报错并退出。

3.2.3. 内存释放过程

假设释放一块大小为4个页面的连续物理内存：

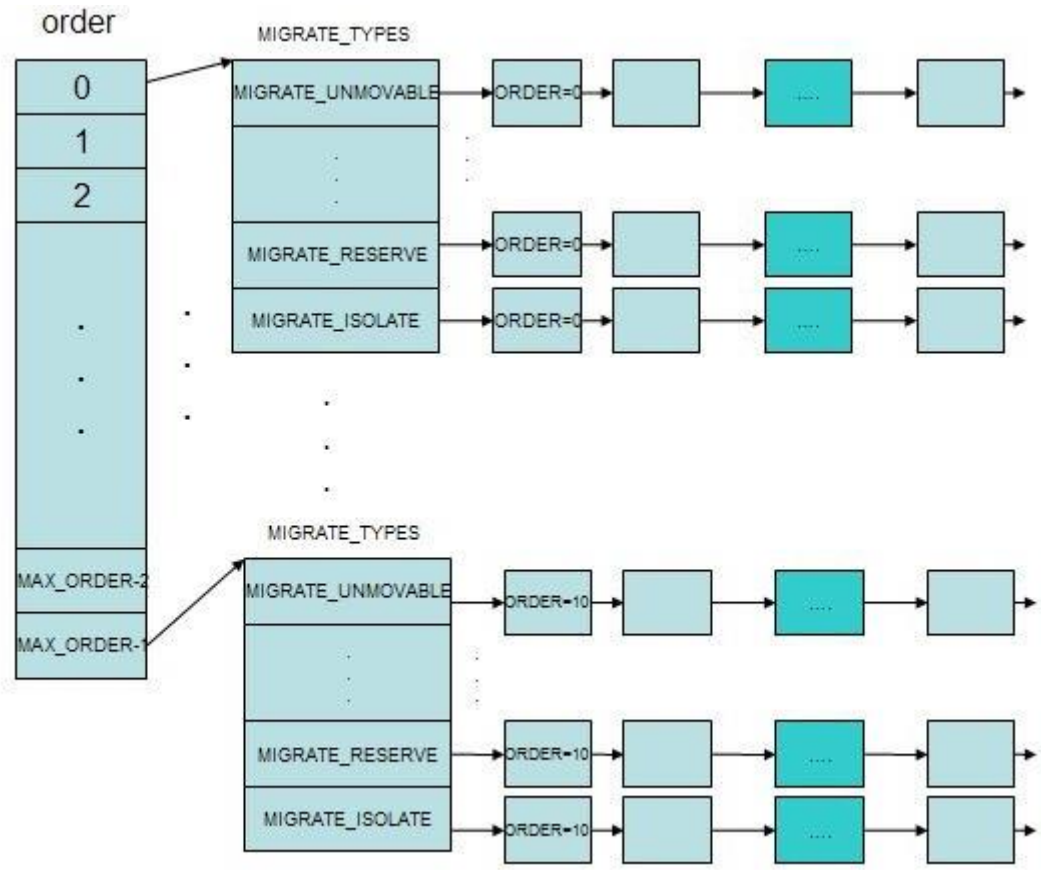
1) 首先检查该块是否有相邻的伙伴块可一起释放；
2) 如果没有，则释放该块，并加入到伙伴算法的空闲链表中（第2个组，2^2=4）；
3) 如果有，则合并。合并之后，再查找是否有相同大小的伙伴块(大小8个页面)可一起释放，依次类推。

3.2.4. 页面迁移

MIGRATE_TYPES指定了迁移类型的种类数。

```
#define MIGRATE_UNMOVABLE     0   //不可移动页：在内存中有固定位置，不能移动到其他地方
#define MIGRATE_RECLAIMABLE   1   //可回收页：不能直接移动，但可以删除，其内容可以从某些源重新生成
#define MIGRATE_MOVABLE       2   //可移动页：可以随意的移动
#define MIGRATE_RESERVE       3   //如果向具有特定可移动性地列表请求分配内存失败，这种紧急情况下可以从MIGRATE_RESERVE分配内存
#define MIGRATE_ISOLATE       4   //是一个特殊的虚拟区域，用于跨域NUMA结点移动物理内存页，在大型系统上，它有益于将物理内存页移动到接近于是用该页最频繁的CPU
#define MIGRATE_TYPES         5   //只是表示迁移类型的数目，不代表具体的区域
```

图 3-8 伙伴算法将页面迁移类型进行分组



4. 内存内碎片

4.1. 内碎片概要

1) 内部碎片是位于内存页面内部的存储空间。
2) 占有这些区域或页面的进程并不使用这个存储块。而在进程占有这块存储块时，系统无法利用它。
3) 直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

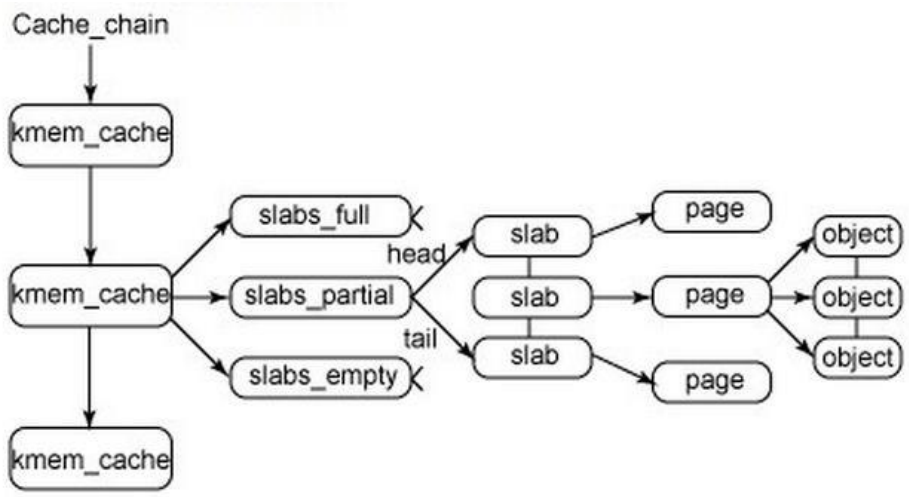
伙伴算法分配内存时，每次至少分配一个页面(4KB)。
当请求的内存大小为几十个字节或几百个字节时应该如何处理？
内碎片又如何解决？

4.2. slab分配器

4.2.1. slab分配器概要

slab分配器工作在伙伴系统的上层。它基于对象进行管理，将相同类型的对象归为一类(如进程描述符就是一类)。每当要申请这样一个对象，slab分配器就从一个slab列表中分配一个这样大小的单元出去;而当要释放时，将其重新保存在该列表中，而不是直接返回给伙伴系统。

图 4-9 slab分配器的结构



注：slab分配器所使用的缓存是内存中的缓存，而不是CPU cache。

为什么使用slab分配器？

1) 内核通常依赖于对小对象的分配，它们会在系统生命周期内进行无数次分配。slab分配器通过对类似大小的对象进行缓存，避免了常见的内碎片问题。
2) slab分配器还支持通用对象的初始化，从而避免了为同一对象重复进行初始化。

4.2.2. slabtop

```
# slabtop -d 1
Active / Total Objects (% used)    : 551769 / 609815 (90.5%)
 Active / Total Slabs (% used)      : 8847 / 8847 (100.0%)
 Active / Total Caches (% used)     : 70 / 110 (63.6%)
 Active / Total Size (% used)       : 44541.33K / 57160.78K (77.9%)
 Minimum / Average / Maximum Object : 0.01K / 0.09K / 15.69K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
156160 154998  99%    0.03K   1220      128      4880K kmalloc-32
115260 115260 100%    0.02K    678      170      2712K fsnotify_event_holder
 76288  76288 100%    0.01K    149      512       596K kmalloc-8
 52736  52736 100%    0.02K    206      256       824K kmalloc-16
```

```
 45440  26095  57%    0.06K    710       64      2840K kmalloc-64
 28050  17024  60%    0.08K    550       51      2200K selinux_inode_security
 24129  15004  62%    0.19K   1149       21      4596K dentry
 19008  18904  99%    0.11K    528       36      2112K sysfs_dir_cache
 11988  10956  91%    0.58K    444       27      7104K inode_cache
 11628  11550  99%    0.21K    646       18      2584K vm_area_struct
  8848   2442  27%    0.57K    316       28      5056K radix_tree_node
  7098   6182  87%    0.09K    169       42       676K kmalloc-96
  5552   5003  90%    0.25K    347       16      1388K kmalloc-256
  5440   5128  94%    0.06K     85       64       340K anon_vma
  4256   4256 100%    0.07K     76       56       304K Acpi-ParseExt
  4212   1254  29%    0.15K    162       26       648K xfs_ili
  4000   2654  66%    0.12K    125       32       500K kmalloc-128
  3744   1449  38%    1.00K    234       16      3744K xfs_inode
  3230   3230 100%    0.05K     38       85       152K shared_policy_node
  2772   2772 100%    0.19K    132       21       528K kmalloc-192
  2184   2068  94%    0.10K     56       39       224K buffer_head
  2142   2142 100%    0.04K     21      102        84K Acpi-Namespace
  1648   1539  93%    1.00K    103       16      1648K kmalloc-1024
  1464   1240  84%    0.66K     61       24       976K shmem_inode_cache
  1224   1146  93%    0.12K     36       34       144K fsnotify_event
  1152    904  78%    0.50K     72       16       576K kmalloc-512
  1080    980  90%    0.64K     45       24       720K proc_inode_cache
   704    586  83%    2.00K     44       16      1408K kmalloc-2048
   665    532  80%    0.81K     35       19       560K task_xstate
   616    616 100%    1.12K     22       28       704K signal_cache
   588    588 100%    0.38K     28       21       224K blkdev_requests
   527    419  79%    0.23K     31       17       124K cfq_queue
   525    525 100%    0.62K     21       25       336K sock_inode_cache
   476    476 100%    0.94K     28       17       448K RAW
   396    396 100%    0.44K     22       18       176K scsi_cmd_cache
   360    360 100%    0.39K     18       20       144K xfs_efd_item
   352    326  92%    2.84K     32       11      1024K task_struct
   352    330  93%    0.18K     16       22        64K xfs_log_ticket
   330    302  91%    2.06K     22       15       704K sighand_cache
   320    320 100%    0.06K      5       64        20K kmem_cache_node
   292    292 100%    0.05K      4       73        16K ip_fib_trie
...
#
```

5. 内存相关的性能评价标准

No.	场景	说明
1	memset	评价申请物理内存的速度。
2	memcpy	评价物理内存间拷贝的速度。
3	跨node内存访问	评价跨node访问时的memory性能。
4	memory overcommit	通过swap等页面置换方法，实现memory overcommit。评价该场合下的内存性能。