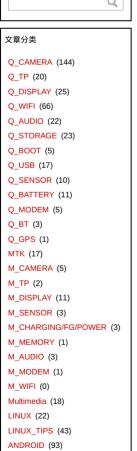
登录 | 注册

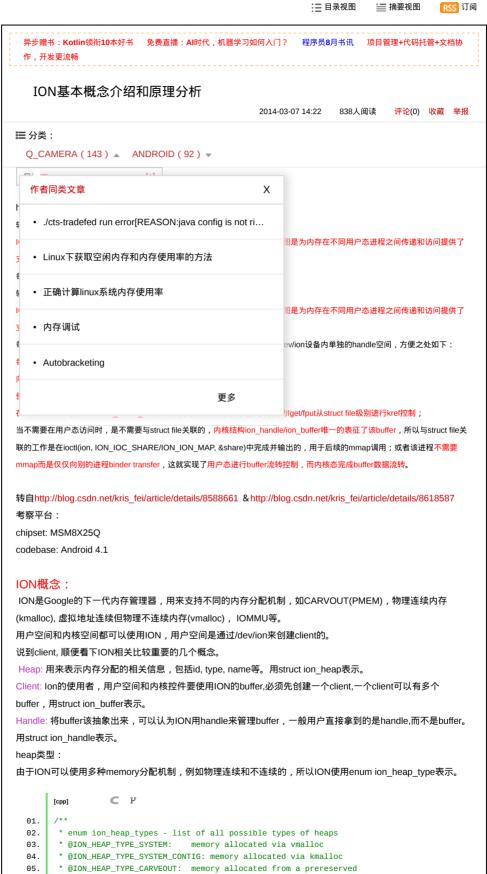
### Gabby





DRIVER (14)

**APK** (25)



第1页 共69页 2017年09月07日 04:09

```
LINUX_PRO (6)
OS Install Tips (12)
C++ (60)
HR (9)
health (1)
tv&&music&&fun (9)
compile (9)
并发处理相关 (8)
const&mutable (2)
callback_func (1)
ipc (1)
camera_framework (9)
sp&wp (3)
selinux (1)
initro (3)
cmd (2)
virtual_func (2)
stl (1)
log (2)
mediaplayer (1)
parcel (1)
property (2)
debug (1)
busybox (1)
bionic (1)
zygote (2)
surface (2)
jni (1)
笔试题目 (3)
平台设备总线模型 (1)
字符驱动 (1)
```

```
文章存档

2016年04月 (32)
2016年03月 (6)
2016年02月 (13)
2015年12月 (43)
2015年11月 (9)
展开
```

```
阅读排行
请教:过量喝咖啡对心脏 (4385)
C++和JAVA的区别 - 给礼 (3934)
svn如何取消认证缓存设1 (3792)
android TP (2739)
Android LCD (2562)
高通平台android开发总约 (2280)
Android图形合成和显示到 (2258)
一个离职员工对中兴的回 (2237)
高通QPST Download使F (2208)
android camera (2088)
```

```
推荐文章

* CSDN日报20170828——(4个方法快速打造你的阅读清单)
* Android检查更新下载安装
* 动手打造史上最简单的
Recycleview 侧滑菜单

* TCP网络通讯如何解决分包粘包问题
* 程序员的八重境界
```

```
06
                       carveout heap, allocations are physically
07.
08.
       * @ION_HEAP_TYPE_IOMMU: IOMMU memory
09.
        @ION HEAP TYPE CP:
                             memory allocated from a prereserved
10.
                      carveout heap, allocations are physically
11.
                      contiguous. Used for content protection.
      * @ION_HEAP_TYPE_DMA:
12.
                                      memory allocated via DMA API
      * @ION_HEAP_END:
                             helper for iterating over heaps
13.
14.
15.
      enum ion heap type {
         ION_HEAP_TYPE_SYSTEM,
16.
17.
          ION_HEAP_TYPE_SYSTEM_CONTIG,
18.
          ION_HEAP_TYPE_CARVEOUT,
19.
          ION_HEAP_TYPE_IOMMU,
20.
          ION_HEAP_TYPE_CP,
21.
          ION_HEAP_TYPE_DMA,
          ION_HEAP_TYPE_CUSTOM, /* must be last so device specific heaps always
22.
23.
                       are at the end of this enum */
24.
          ION NUM HEAPS,
25. };
01.
      <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
        enum ion heap types - list of all possible types of heaps
02.
        @ION HEAP TYPE SYSTEM: memory allocated via ymalloc
03.
       * @ION_HEAP_TYPE_SYSTEM_CONTIG: memory allocated via kmalloc
04.
05.
       * @ION_HEAP_TYPE_CARVEOUT: memory allocated from a prereserved
06.
                       carveout heap, allocations are physically
07.
                       contiquous
      * @ION_HEAP_TYPE_IOMMU: IOMMU memory
08.
       * @ION_HEAP_TYPE_CP:
09.
                              memory allocated from a prereserved
10.
                      carveout heap, allocations are physically
11.
                      contiguous. Used for content protection.
12.
      * @ION_HEAP_TYPE_DMA:
                                      memory allocated via DMA API
13.
       * @ION_HEAP_END:
                              helper for iterating over heaps
14.
      enum ion_heap_type {
15.
16.
         ION_HEAP_TYPE_SYSTEM,
17.
          ION_HEAP_TYPE_SYSTEM_CONTIG,
          ION_HEAP_TYPE_CARVEOUT,
18.
19.
          ION_HEAP_TYPE_IOMMU,
20.
          ION_HEAP_TYPE_CP,
          ION HEAP TYPE DMA
21.
          ION_HEAP_TYPE_CUSTOM, /* must be last so device specific heaps always
22.
23.
                       are at the end of this enum ^{\star}/
          ION_NUM_HEAPS,
24.
25. };</span>
```

代码中的注释很明确地说明了哪种type对应的是分配哪种memory。不同type的heap需要不同的method去分配,不过都是用struct ion\_heap\_ops来表示的。如以下例子:

```
[cpp]
01.
      static struct ion_heap_ops carveout_heap_ops = {
02.
          .allocate = ion_carveout_heap_allocate,
03
          .free = ion_carveout_heap_free,
04.
          .phys = ion_carveout_heap_phys,
05.
          .map user = ion carveout heap map user,
06.
          .map kernel = ion carveout heap map kernel,
07.
          .unmap_user = ion_carveout_heap_unmap_user,
08
          .unmap_kernel = ion_carveout_heap_unmap_kernel,
09.
          .map_dma = ion_carveout_heap_map_dma
                                                                                                     关闭
10.
          .unmap_dma = ion_carveout_heap_unmap_dma,
11.
          .cache op = ion carveout cache ops,
12.
          .print_debug = ion_carveout_print_debug,
13.
          .map_iommu = ion_carveout_heap_map_iommu,
14.
          .unmap_iommu = ion_carveout_heap_unmap_iommu,
15.
16.
      static struct ion_heap_ops kmalloc_ops = {
17.
18.
          .allocate = ion system contig heap allocate,
19.
          .free = ion_system_contig_heap_free,
20.
          .phys = ion_system_contig_heap_phys,
21.
          .map_dma = ion_system_contig_heap_map_dma,
          .unmap_dma = ion_system_heap_unmap_dma,
23.
          .map kernel = ion system heap map kernel
```

第2页 共69页 2017年09月07日 04:09

\* 四大线程池详解

```
24.
          .unmap_kernel = ion_system_heap_unmap_kernel,
25.
          .map_user = ion_system_contig_heap_map_user,
26.
          .cache_op = ion_system_contig_heap_cache_ops,
27.
          .print debug = ion system contig print debug.
28.
          .map iommu = ion_system_contig_heap_map_iommu,
29.
          .unmap_iommu = ion_system_heap_unmap_iommu,
30.
01.
      <span xmlns="http://www.w3.org</pre>
      /1999/xhtml" style="">static struct ion_heap_ops carveout_heap_ops = {
02.
          .allocate = ion carveout heap allocate,
03.
          .free = ion carveout heap free.
04.
          .phys = ion_carveout_heap_phys,
05.
          .map_user = ion_carveout_heap_map_user,
          .map_kernel = ion_carveout_heap_map_kernel,
07.
          .unmap_user = ion_carveout_heap_unmap_user,
08.
          .unmap_kernel = ion_carveout_heap_unmap_kernel,
09.
          .map dma = ion carveout heap map dma.
10.
          .unmap dma = ion carveout heap unmap dma
11.
          .cache_op = ion_carveout_cache_ops,
12.
          .print_debug = ion_carveout_print_debug,
13.
          .map_iommu = ion_carveout_heap_map_iommu,
14.
          .unmap iommu = ion carveout heap unmap iommu.
15.
16.
17.
      static struct ion_heap_ops kmalloc_ops = {
         .allocate = ion_system_contig_heap_allocate,
18.
19.
          .free = ion_system_contig_heap_free,
          .phys = ion system contig heap phys.
20.
21.
          .map dma = ion system contig heap map dma,
22.
          .unmap_dma = ion_system_heap_unmap_dma,
23.
          .map_kernel = ion_system_heap_map_kernel,
24.
          .unmap_kernel = ion_system_heap_unmap_kernel,
          .map_user = ion_system_contig_heap_map_user,
26.
          .cache op = ion system contig heap cache ops.
27.
          .print debug = ion system contig print debug,
28.
          .map_iommu = ion_system_contig_heap_map_iommu,
29.
          .unmap_iommu = ion_system_heap_unmap_iommu,
30. };</span>
```

#### Heap ID

同一种type的heap上当然可以分为若该干个chunk供用户使用,所以ION又使用ID来区分了。例如在type为ION\_HEAP\_TYPE\_CARVEOUT的heap上,audio和display部分都需要使用,ION就用ID来区分。 Heap id用enumion\_heap\_ids表示。

```
[qqɔ]
01.
02.
       ^{\star} These are the only ids that should be used for Ion heap ids.
      * The ids listed are the order in which allocation will be attempted
       ^{\ast} if specified. Don't swap the order of heap ids unless you know what
05.
      * you are doing!
       ^{\star} Id's are spaced by purpose to allow new Id's to be inserted in-between (for
06.
       * possible fallbacks)
07.
08.
09.
10.
      enum ion_heap_ids {
          INVALID_HEAP_ID = -1,
11.
          ION CP MM HEAP ID = 8.
12.
13.
          ION CP MFC HEAP ID = 12,
                                                                                                       关闭
          ION\_CP\_WB\_HEAP\_ID = 16, /* 8660 only */
14.
          ION\_CAMERA\_HEAP\_ID = 20, /* 8660 only */
15.
          ION\_SF\_HEAP\_ID = 24,
17.
          ION_IOMMU_HEAP_ID = 25,
18.
          ION OSECOM HEAP ID = 26.
19.
          ION AUDIO HEAP BL ID = 27,
20.
          ION_AUDIO_HEAP_ID = 28,
21.
22.
          ION_MM_FIRMWARE_HEAP_ID = 29,
23.
          ION_SYSTEM_HEAP_ID = 30,
24.
          ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_SECURE flag */
25.
26. };
```

第3页 共69页 2017年09月07日 04:09

```
[cpp]
  01.
        <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
         * These are the only ids that should be used for Ion heap ids.
  02.
        * The ids listed are the order in which allocation will be attempted
  03.
         * if specified. Don't swap the order of heap ids unless you know what
        * you are doing!
        * Id's are spaced by purpose to allow new Id's to be inserted in-between (for
  06.
        * possible fallbacks)
  07.
  08.
  09.
  10.
        enum ion_heap_ids {
  11.
            INVALID_HEAP_ID = -1,
            ION_CP_MM_HEAP_ID = 8,
  12.
            ION CP MFC HEAP ID = 12,
  13.
           ION_CP_WB_HEAP_ID = 16, /* 8660 only */
  14.
  15.
            ION\_CAMERA\_HEAP\_ID = 20, /* 8660 only */
  16.
            ION_SF_HEAP_ID = 24,
            ION_IOMMU_HEAP_ID = 25,
            ION_QSECOM_HEAP_ID = 26,
  18.
            ION AUDIO HEAP BL ID = 27.
  19.
  20.
            ION AUDIO HEAP ID = 28,
  21.
  22.
            ION_MM_FIRMWARE_HEAP_ID = 29,
  23.
            ION_SYSTEM_HEAP_ID = 30,
  24.
            ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_SECURE flag */
  25.
  26. };</span>
Heap 定义:
    了解了heaptype和id,看看如何被用到了,本平台使用的文件为board-qrd7627a.c,有如下定义:
                   CP
        [cpp]
  02.
         * These heaps are listed in the order they will be allocated.
        * Don't swap the order unless you know what you are doing!
  03.
  04.
  05.
        struct ion_platform_heap msm7627a_heaps[] = {
  06.
               {
  07.
                    .id = ION_SYSTEM_HEAP_ID,
  08.
                    .type = ION HEAP TYPE SYSTEM.
  09.
                    .name = ION VMALLOC HEAP NAME,
  10.
               },
  11.
        #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
  12.
                /* PMEM_ADSP = CAMERA */
  13.
               {
                    .id = ION CAMERA HEAP ID.
  14.
                   .type = CAMERA_HEAP_TYPE,
  15.
  16.
                    .name = ION CAMERA HEAP NAME,
  17.
                    .memory_type = ION_EBI_TYPE,
  18.
                    .extra_data = (void *)&co_mm_ion_pdata,
  19.
                    .priv = (void *)&ion_cma_device.dev,
  20.
                /* AUDIO HEAP 1*/
  21.
  22.
               {
  23.
                    .id = ION_AUDIO_HEAP_ID,
  24.
                   .type = ION_HEAP_TYPE_CARVEOUT,
                   .name = ION_AUDIO_HEAP_NAME,
                    .memory_type = ION_EBI_TYPE,
  26.
                    .extra_data = (void *)&co_ion_pdata,
  27.
  28.
               },
  29.
                /* PMEM_MDP = SF */
                                                                                                   关闭
  30.
  31.
                    .id = ION_SF_HEAP_ID,
                   .type = ION_HEAP_TYPE_CARVEOUT,
  32.
  33.
                    .name = ION SF HEAP NAME,
  34.
                    .memory_type = ION_EBI_TYPE,
  35.
                    .extra_data = (void *)&co_ion_pdata,
  36.
               /* AUDIO HEAP 2*/
  38.
               {
                    .id = ION AUDIO HEAP BL ID.
  39.
  40.
                    .type = ION_HEAP_TYPE_CARVEOUT,
  41.
                    .name = ION_AUDIO_BL_HEAP_NAME,
                   .memory_type = ION_EBI_TYPE,
```

第4页 共69页 2017年09月07日 04:09

43.

.extra\_data = (void \*)&co\_ion\_pdata,

```
44.
                  .base = BOOTLOADER_BASE_ADDR,
45.
46.
47.
     #endif
48.
     };
      <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
Θ1
02.
       * These heaps are listed in the order they will be allocated.
03.
      * Don't swap the order unless you know what you are doing!
05.
     struct ion_platform_heap msm7627a_heaps[] = {
06.
07.
                  .id = ION SYSTEM HEAP ID.
08.
                  .type = ION_HEAP_TYPE_SYSTEM,
09.
                  .name = ION_VMALLOC_HEAP_NAME,
10.
11.
     #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
             /* PMEM ADSP = CAMERA */
12.
13.
14.
                  .id = ION_CAMERA_HEAP_ID,
15.
                  .type = CAMERA_HEAP_TYPE,
16.
                 .name = ION_CAMERA_HEAP_NAME,
17.
                  .memory type = ION EBI TYPE,
                  .extra_data = (void *)&co_mm_ion_pdata,
18.
19.
                  .priv = (void *)&ion_cma_device.dev,
20.
21.
             /* AUDIO HEAP 1*/
22.
             {
                  .id = ION AUDIO HEAP ID.
23.
                 .type = ION_HEAP_TYPE_CARVEOUT,
.name = ION_AUDIO_HEAP_NAME,
24.
25.
26.
                  .memory_type = ION_EBI_TYPE,
27.
                  .extra_data = (void *)&co_ion_pdata,
28.
             },
             /* PMEM_MDP = SF */
29.
             {
30.
31.
                  .id = ION SF HEAP ID,
32.
                  .type = ION_HEAP_TYPE_CARVEOUT,
33.
                  .name = ION_SF_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_ion_pdata,
35.
36.
             },
             /* AUDIO HEAP 2*/
37.
38.
39.
                  .id = ION_AUDIO_HEAP_BL_ID,
                 .type = ION_HEAP_TYPE_CARVEOUT,
41.
                  .name = ION_AUDIO_BL_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
42.
                  .extra_data = (void *)&co_ion_pdata,
43.
44.
                  .base = BOOTLOADER_BASE_ADDR,
45.
46.
     #endif
47.
48. };</span>
```

#### ION Handle:

当Ion client分配buffer时,相应的一个唯一的handle也会被指定,当然client可以多次申请ion buffer。申请好buffer之后,返回的是一个ion handle,不过要知道Ion buffer才和实际的内存相关,包括size, address等信息。Struct ion\_handle和struct ion\_buffer如下:

```
关闭
                 CP
02.
      * ion_handle - a client local reference to a buffer
      * @ref:
03.
                    reference count
      * @client:
04.
                    back pointer to the client the buffer resides in
05.
      * @buffer:
                     pointer to the buffer
                     node in the client's handle rbtree
      * @kmap_cnt:
                     count of times this client has mapped to kernel
08.
      * @dmap_cnt:
                        count of times this client has mapped for dma
09.
10.
      ^{\star} Modifications to node, map_cnt or mapping should be protected by the
      ^{\star} lock in the client. Other fields are never changed after initialization.
11.
12.
13. struct ion_handle {
```

第5页 共69页 2017年09月07日 04:09

```
14.
          struct kref ref;
15.
          struct ion_client *client;
          struct ion_buffer *buffer;
16.
17.
         struct rb node node:
18.
         unsigned int kmap_cnt;
19.
          unsigned int iommu_map_cnt;
20.
21.
22.
23.
      * struct ion_buffer - metadata for a particular buffer
      * @ref:
24.
                     refernce count
25.
       * @node:
                     node in the ion_device buffers tree
26.
      * @dev:
                     back pointer to the ion_device
                     back pointer to the heap the buffer came from
      * @flags: buffer specific flags
29.
      * @size:
                     size of the buffer
       * @priv_virt:
30.
                        private data to the buffer representable as
31.
                a void *
32.
      * @priv_phys:
                        private data to the buffer representable as
               an ion_phys_addr_t (and someday a phys_addr_t)
34.
      * @lock:
                     protects the buffers cnt fields
       * @kmap_cnt:
35.
                         number of times the buffer is mapped to the kernel
       * @vaddr:
36.
                     the kenrel mapping if kmap_cnt is not zero
37.
       * @dmap_cnt:
                       number of times the buffer is mapped for dma
      * @sg_table:
38.
                        the sg table for the buffer if dmap_cnt is not zero
39.
40.
     struct ion_buffer {
41.
         struct kref ref;
42.
          struct rb_node node;
43.
          struct ion_device *dev;
44.
          struct ion_heap *heap;
45.
          unsigned long flags;
46.
         size_t size;
47.
         union {
             void *priv virt;
48.
49.
             ion_phys_addr_t priv_phys;
50.
          struct mutex lock;
52.
         int kmap_cnt;
53.
          void *vaddr;
54.
         int dmap_cnt;
55.
          struct sg_table *sg_table;
56.
57.
          unsigned int iommu_map_cnt;
58.
          struct rb_root iommu_maps;
59.
          int marked;
60. };
      <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
01.
02.
       ^{\star} ion_handle - a client local reference to a buffer
      * @ref:
                     reference count
      * @client:
                     back pointer to the client the buffer resides in
       * @buffer: pointer to the buffer
05.
       * @node:
06.
                     node in the client's handle rbtree
07.
      * @kmap_cnt:
                        count of times this client has mapped to kernel
08.
                        count of times this client has mapped for dma
      ^{\star} Modifications to node, map_cnt or mapping should be protected by the
      * lock in the client. Other fields are never changed after initialization.
11.
12.
13.
      struct ion_handle {
14.
          struct kref ref;
          struct ion_client *client;
16.
          struct ion_buffer *buffer;
                                                                                                  关闭
17.
         struct rb_node node;
18.
         unsigned int kmap_cnt;
19.
          unsigned int iommu_map_cnt;
20.
21.
      * struct ion_buffer - metadata for a particular buffer
23.
       * @ref:
24.
                    refernce count
       * @node:
25.
                     node in the ion_device buffers tree
26.
      * @dev:
                     back pointer to the ion_device
                    back pointer to the heap the buffer came from
27.
      * @flags: buffer specific flags
* @size: size of the buffer
28.
29.
      * @priv_virt:
30.
                        private data to the buffer representable as
```

第6页 共69页 2017年09月07日 04:09

```
31.
                   a void *
  32.
        * @priv_phys:
                           private data to the buffer representable as
  33.
                  an ion_phys_addr_t (and someday a phys_addr_t)
        * @lock:
                      protects the buffers cnt fields
  34.
        * @kmap_cnt:
                         number of times the buffer is mapped to the kernel
  35.
        * @vaddr: the kenrel mapping if kmap_cnt is not zero
* @dmap_cnt: number of times the buffer is mapped for
  36.
  37.
                          number of times the buffer is mapped for dma
  38.
        * @sg_table:
                         the sg table for the buffer if dmap_cnt is not zero
  39.
  40.
       struct ion_buffer {
  41.
           struct kref ref:
  42.
            struct rb_node node;
  43.
            struct ion_device *dev;
  44.
            struct ion_heap *heap;
  45.
           unsigned long flags;
  46.
           size_t size;
  47.
           union {
  48.
               void *priv_virt;
  49
               ion_phys_addr_t priv_phys;
  50.
  51.
            struct mutex lock;
           int kmap_cnt;
  52.
           void *vaddr:
  53.
  54.
           int dmap_cnt;
  55.
            struct sg_table *sg_table;
  56.
            int umap_cnt;
           unsigned int iommu_map_cnt;
            struct rb root iommu maps;
  58.
           int marked:
  59.
  60. };</span>
ION Client:
    用户空间和内核空间都可以成为client,不过创建的方法稍稍有点区别,先了解下基本的操作流程吧。
内核空间:
先创建client:
                   C Y
       struct ion_client *ion_client_create(struct ion_device *dev,
  01.
  02.
                            unsigned int heap mask,
                            const char *name)
  03.
       [cpp]
  01.
        <span xmlns="http://www.w3.org</pre>
        /1999/xhtml" style="">struct ion_client *ion_client_create(struct ion_device *dev,
  02.
                            unsigned int heap_mask,
  03.
                            const char *name)
heap_mask: 可以分配的heap type, 如carveout,system heap, iommu等。
高通使用msm ion client create函数封装了下。
有了client之后就可以分配内存:
                   C P
       [cpp]
        struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
  01.
  02.
                        size_t align, unsigned int flags)
        <span xmlns="http://www.w3.org</pre>
                                                                                                  关闭
        /1999/xhtml" style="">struct ion_handle *ion_alloc(st
  02.
                        size_t align, unsigned int flags)
flags: 分配的heap id.
有了handle也就是buffer之后就准备使用了,不过还是物理地址,需要map:
                   CP
        void *ion_map_kernel(struct ion_client *client, struct ion_handle *handle,
  01.
  02.
                   unsigned long flags)
       [qqɔ]
       <span xmlns="http://www.w3.org</pre>
```

第7页 共69页 2017年09月07日 04:09

```
/1999/xhtml" style="">void *ion_map_kernel(struct ion_client *client, struct ion_handle
  02.
                  unsigned long flags)</span>
用户空间
    用户空间如果想使用ION,也必须先要创建client,不过它是打开/dev/ion,实际上它最终也会调用
    不过和内核空间创建client的一点区别是,用户空间不能选择heap type(使用预订的heap id隐含heap
type),但是内核空间却可以。
    另外,用户空间是通过IOCTL来分配内存的,cmd为ION_IOC_ALLOC.
                  C P
       [cpp]
      ion_fd = open("/dev/ion", O_ RDONLY | O_SYNC);
  01.
  02. ioctl(ion_fd, ION_IOC_ALLOC, alloc);
       <span xmlns="http://www.w3.org/1999/xhtml" style="">ion_fd = open("
       /dev/ion", 0_ RDONLY | 0_SYNC);
  02. ioctl(ion_fd, ION_IOC_ALLOC, alloc); </span>
alloc为struct ion_allocation_data,len是申请buffer的长度,flags是heap id。
                  C P
       [cpp]
  01.
       * struct ion_allocation_data - metadata passed from userspace for allocations
  02.
       * @len:
                size of the allocation
       * @align: required alignment of the allocation
       * @flags: flags passed to heap
  05.
  06.
        ^{\star} @handle: pointer that will be populated with a cookie to use to refer
  07.
              to this allocation
  08.
       * Provided by userspace as an argument to the ioctl
  10.
 11.
       struct ion allocation data {
  12.
          size_t len;
 13.
          size_t align;
  14.
          unsigned int flags;
          struct ion_handle *handle;
  15.
  16. };
       <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
  01.
  02.
        * struct ion_allocation_data - metadata passed from userspace for allocations
       * @len: size of the allocation
  03.
  04.
       * @align: required alignment of the allocation
       * @flags: flags passed to heap
       * @handle: pointer that will be populated with a cookie to use to refer
  07.
             to this allocation
  08.
       * Provided by userspace as an argument to the ioctl
  09.
  10.
  11.
      struct ion_allocation_data {
  12.
          size t len;
 13.
          size t align:
          unsigned int flags;
 14.
  15.
          struct ion_handle *handle;
                                                                                          关闭
分配好了buffer之后,如果用户空间想使用buffer,先需要mmap. ION是過過ル場用してLT可
ION_IOC_SHARE/ION_IOC_MAP来得到可以mmap的fd,然后再执行mmap得到bufferaddress.
然后,你也可以将此fd传给另一个进程,如通过binder传递。在另一个进程中通过ION IOC IMPORT这个IOCTL
来得到这块共享buffer了。
来看一个例子:
                 C P
       [cpp]
  01.
  02.
       int ionfd = open("/dev/ion", 0_RDONLY | 0_DSYNC);
       alloc_data.len = 0x1000;
  03.
  04. alloc_data.align = 0x1000;
```

第8页 共69页 2017年09月07日 04:09

```
05.
     alloc_data.flags = ION_HEAP(ION_CP_MM_HEAP_ID);
06.
     rc = ioctl(ionfd,ION_IOC_ALLOC, &alloc_data);
     fd_data.handle = alloc_data.handle;
08.
     rc = ioctl(ionfd, ION_IOC_SHARE, &fd_data);
09.
     shared_fd = fd_data.fd;
10.
11.
     fd_data.fd = shared_fd;
     rc = ioctl(ionfd, ION_IOC_IMPORT, &fd_data);
01.
     <span xmlns="http://www.w3.org/1999/xhtml" style="">进程A:
     int ionfd = open("/dev/ion", 0 RDONLY | 0 DSYNC);
02.
03.
     alloc data.len = 0x1000;
04.
     alloc_data.align = 0x1000;
     alloc_data.flags = ION_HEAP(ION_CP_MM_HEAP_ID);
     rc = ioctl(ionfd, ION_IOC_ALLOC, &alloc_data);
     fd_data.handle = alloc_data.handle;
     rc = ioctl(ionfd.ION IOC SHARE.&fd data):
08.
09.
     shared fd = fd data.fd;
10.
11.
     进程B:
     fd_data.fd = shared_fd;
12.
13. rc = ioctl(ionfd, ION_IOC_IMPORT, &fd_data); </span>
```

从上一篇ION基本概念中,我们了解了heaptype, heap id, client, handle以及如何使用,本篇再从原理上分析下ION的运作流程

MSM8x25Q平台使用的是board-qrd7627.c,ION相关定义如下:

```
CP
     [cpp]
01.
      * These heaps are listed in the order they will be allocated.
      * Don't swap the order unless you know what you are doing!
03.
04.
05.
     struct ion_platform_heap msm7627a_heaps[] = {
06.
07.
                  .id = ION_SYSTEM_HEAP_ID,
                 .type = ION_HEAP_TYPE_SYSTEM,
09.
                 .name = ION_VMALLOC_HEAP_NAME,
10.
     #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
11.
12.
             /* PMEM_ADSP = CAMERA */
             {
                  .id = ION_CAMERA_HEAP_ID,
                 .type = CAMERA_HEAP_TYPE,
15.
                 .name = ION CAMERA HEAP NAME,
16.
17.
                 .memory_type = ION_EBI_TYPE,
18.
                 .extra_data = (void *)&co_mm_ion_pdata,
19.
                 .priv = (void *)&ion_cma_device.dev,
             },
21.
             /* AUDIO HEAP 1*/
22.
             {
23.
                  .id = ION AUDIO HEAP ID,
24.
                 .type = ION_HEAP_TYPE_CARVEOUT,
25.
                 .name = ION_AUDIO_HEAP_NAME,
                 .memory_type = ION_EBI_TYPE,
26.
                 .extra_data = (void *)&co_ion_pdata,
28.
             },
             /* PMEM_MDP = SF */
29.
                                                                                                 关闭
30.
31.
                 .id = ION_SF_HEAP_ID,
                 .type = ION_HEAP_TYPE_CARVEOUT,
33.
                 .name = ION_SF_HEAP_NAME,
34.
                 .memory_type = ION_EBI_TYPE,
                 .extra_data = (void *)&co_ion_pdata,
35.
36.
37.
             /* AUDIO HEAP 2*/
38.
             {
39.
                 .id = ION_AUDIO_HEAP_BL_ID,
                 .type = ION_HEAP_TYPE_CARVEOUT,
40.
41.
                 .name = ION_AUDIO_BL_HEAP_NAME,
42.
                 .memory_type = ION_EBI_TYPE,
43.
                 .extra_data = (void *)&co_ion_pdata,
                 .base = BOOTLOADER_BASE_ADDR,
```

第9页 共69页 2017年09月07日 04:09

```
45.
             },
46.
47.
     #endif
48.
49.
50.
      static struct ion_co_heap_pdata co_ion_pdata = {
51.
         .adjacent_mem_id = INVALID_HEAP_ID,
         .align = PAGE_SIZE,
52.
53.
54.
55.
      static struct ion_co_heap_pdata co_mm_ion_pdata = {
56.
          .adjacent_mem_id = INVALID_HEAP_ID,
57.
          .align = PAGE_SIZE,
58.
59.
60.
     static u64 msm_dmamask = DMA_BIT_MASK(32);
61.
62.
      static struct platform_device ion_cma_device = {
63.
         .name = "ion-cma-device",
64.
         .id = -1,
65.
         .dev = {
            .dma_mask = &msm_dmamask,
66.
             .coherent_dma_mask = DMA_BIT_MASK(32),
67.
68.
69. };
01.
      <span xmlns="http://www.w3.org/1999/xhtml" style="">/**
       ^{\star} These heaps are listed in the order they will be allocated.
       ^{\star} Don't swap the order unless you know what you are doing!
03.
04.
05.
     struct ion_platform_heap msm7627a_heaps[] = {
06.
07.
                  .id = ION_SYSTEM_HEAP_ID,
                 .type = ION_HEAP_TYPE_SYSTEM,
09.
                  .name = ION VMALLOC HEAP NAME.
10.
     #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
11.
12.
             /* PMEM_ADSP = CAMERA */
13.
14.
                  .id = ION_CAMERA_HEAP_ID,
                 .type = CAMERA_HEAP_TYPE,
15.
                  .name = ION CAMERA HEAP NAME,
16.
17.
                  .memory\_type = ION\_EBI\_TYPE,
18.
                  .extra_data = (void *)&co_mm_ion_pdata,
19.
                  .priv = (void *)&ion_cma_device.dev,
             },
21.
             /* AUDIO HEAP 1*/
22.
23.
                  .id = ION_AUDIO_HEAP_ID,
24.
                  .type = ION_HEAP_TYPE_CARVEOUT,
25.
                 .name = ION_AUDIO_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_ion_pdata,
27.
             },
/* PMEM_MDP = SF */
28.
29.
30.
31.
                  .id = ION_SF_HEAP_ID,
                 .type = ION_HEAP_TYPE_CARVEOUT,
33.
                 .name = ION_SF_HEAP_NAME,
34.
                  .memory_type = ION_EBI_TYPE,
35.
                  .extra_data = (void *)&co_ion_pdata,
36.
37.
             /* AUDIO HEAP 2*/
                                                                                                   关闭
             {
                  .id = ION_AUDIO_HEAP_BL_ID,
39.
40.
                 .type = ION_HEAP_TYPE_CARVEOUT,
41.
                  .name = ION AUDIO BL HEAP NAME,
42.
                  .memory_type = ION_EBI_TYPE,
43.
                  .extra_data = (void *)&co_ion_pdata,
                  .base = BOOTLOADER_BASE_ADDR,
45.
             },
46.
47.
     #endif
48.
49.
50.
      static struct ion_co_heap_pdata co_ion_pdata = {
51.
         .adjacent_mem_id = INVALID_HEAP_ID,
          .align = PAGE_SIZE,
52.
```

第10页 共69页 2017年09月07日 04:09

```
53.
       };
  54.
  55.
       static struct ion_co_heap_pdata co_mm_ion_pdata = {
  56.
           .adjacent_mem_id = INVALID_HEAP_ID,
  57.
           .align = PAGE SIZE,
  58.
       };
  59.
  60.
       static u64 msm_dmamask = DMA_BIT_MASK(32);
  61.
  62.
       static struct platform_device ion_cma_device = {
          .name = "ion-cma-device",
  63.
  64.
           .id = -1,
  65.
          .dev = {
  66.
              .dma_mask = &msm_dmamask,
              .coherent_dma_mask = DMA_BIT_MASK(32),
  68.
  69. };</span>
Qualcomm提示了不要轻易调换顺序,因为后面代码处理是将顺序定死了的,一旦你调换了,代码就无法正常运
另外,本系统中只使用了ION HEAP TYPE CARVEOUT和 ION HEAP TYPE SYSTEM这两种h
对于ION_HEAP_TYPE_CARVEOUT的内存分配,后面将会发现,其实就是之前讲述过的使用mem pool来分配
Platform device如下,在msm_ion.c中用到。
                  CP
       static struct ion_platform_data ion_pdata = {
  02.
          .nr = MSM ION HEAP NUM.
  03.
           .has_outer_cache = 1,
  04.
           .heaps = msm7627a_heaps,
  05.
  07.
       static struct platform_device ion_dev = {
 08.
          .name = "ion-msm",
           .id = 1.
  09.
 10.
           .dev = { .platform_data = &ion_pdata },
       <span xmlns="http://www.w3.org</pre>
       /1999/xhtml" style="">static struct ion_platform_data ion_pdata = {
  02.
          .nr = MSM ION HEAP NUM,
  03.
           .has outer cache = 1,
  04.
           .heaps = msm7627a_heaps,
  05.
      };
  06.
       static struct platform_device ion_dev = {
  07.
  08.
          .name = "ion-msm",
          .id = 1.
  09.
  10.
           .dev = { .platform_data = &ion_pdata },
  11. };</span>
ION初始化
转到msm_ion.c , ion.c的某些函数也被重新封装了下.万事都从设备匹配开始:
                  C P
       [cpp]
  01.
       static struct platform_driver msm_ion_driver = {
          .probe = msm_ion_probe,
                                                                                           关闭
           .remove = msm_ion_remove,
  04.
          .driver = { .name = "ion-msm" }
  05.
       }:
  06.
       static int __init msm_ion_init(void)
  07.
  08.
           /*调用msm_ion_probe */
  09.
           return platform_driver_register(&msm_ion_driver);
 10.
 11.
 12.
       static int msm_ion_probe(struct platform_device *pdev)
  13.
  14.
           /*即board-qrd7627a.c中的ion_pdata */
```

第11页 共69页 2017年09月07日 04:09

struct ion\_platform\_data \*pdata = pdev->dev.platform\_data;

15.

16.

int err;

```
17.
         int i;
18.
19.
         /*heap数量*/
20.
         num_heaps = pdata->nr;
         /*分配struct ion_heap */
21.
22.
         heaps = kcalloc(pdata->nr, sizeof(struct ion_heap *), GFP_KERNEL);
23.
         if (!heaps) {
25.
             err = -ENOMEM;
26.
             goto out;
27.
         -
/*创建节点,最终是/dev/ion,供用户空间操作。*/
28.
29.
         idev = ion_device_create(NULL);
30.
         if (IS_ERR_OR_NULL(idev)) {
             err = PTR_ERR(idev);
32.
             goto freeheaps;
33.
         ^{-}/*最终是根据adjacent_mem_id 是否定义了来分配相邻内存,
34.
35.
     我们没用到,忽略此函数。*/
36.
         msm_ion_heap_fixup(pdata->heaps, num_heaps);
38.
         /* create the heaps as specified in the board file */
39.
         for (i = 0; i < num_heaps; i++) {</pre>
40.
             struct ion_platform_heap *heap_data = &pdata->heaps[i];
41.
             /*分配ion*/
42.
             msm_ion_allocate(heap_data);
43.
             heap_data->has_outer_cache = pdata->has_outer_cache;
44.
             /*创建ion heap。*/
45.
46.
             heaps[i] = ion_heap_create(heap_data);
             if (IS_ERR_OR_NULL(heaps[i])) {
47.
48.
                 heaps[i] = 0;
49.
50.
             } else {
51.
                 if (heap_data->size)
52.
                     pr_info("ION heap %s created at %lx "
53.
                         "with size %x\n", heap_data->name,
54.
                                   heap_data->base,
                                   heap_data->size);
56.
                 else
57.
                     pr_info("ION heap %s created\n",
58.
                                   heap_data->name);
59.
             /*创建的heap添加到idev中,以便后续使用。*/
61.
             ion_device_add_heap(idev, heaps[i]);
62.
         /*检查heap之间是否有重叠部分*/
63.
         check_for_heap_overlap(pdata->heaps, num_heaps);
64.
65.
         platform_set_drvdata(pdev, idev);
66.
         return 0;
67.
68.
     freeheaps:
69.
         kfree(heaps);
70.
     out:
71.
         return err;
72.
73.
     通过ion_device_create创建/dev/ion节点:
74.
75.
     struct ion device *ion device create(long (*custom ioctl)
                          (struct ion_client *client,
76.
77.
                           unsigned int cmd,
78.
                           unsigned long arg))
79.
80.
         struct ion device *idev;
81.
         int ret;
                                                                                                关闭
82.
83.
         idev = kzalloc(sizeof(struct ion_device), GFP_KERNEL);
85.
             return ERR_PTR(-ENOMEM);
         /*是个misc设备*/
86.
87.
         idev->dev.minor = MISC DYNAMIC MINOR;
88.
         /*节点名字为ion*/
89.
         idev->dev.name = "ion";
90.
         /*fops为ion_fops,所以对应ion的操作都会调用ion_fops的函数指针。*/
         idev->dev.fops = &ion_fops;
         idev->dev.parent = NULL;
92.
93.
         ret = misc_register(&idev->dev);
94.
         if (ret) {
95.
             pr_err("ion: failed to register misc device.\n");
             return ERR_PTR(ret);
```

第12页 共69页 2017年09月07日 04:09

```
97.
 98.
           /*创建debugfs目录,路径为/sys/kernel/debug/ion/*/
           idev->debug_root = debugfs_create_dir("ion", NULL);
100.
          if (IS_ERR_OR_NULL(idev->debug_root))
              pr_err("ion: failed to create debug files.\n");
101.
102.
103.
          idev->custom_ioctl = custom_ioctl;
104.
           idev->buffers = RB_ROOT;
105.
           mutex_init(&idev->lock);
106.
          idev->heaps = RB_ROOT;
107.
          idev->clients = RB ROOT:
           /*在ion目录下创建一个check_leaked_fds文件,用来检查Ion的使用是否有内存泄漏。如果申请了ion之后
108.
       不需要使用却没有释放,就会导致memory leak.*/
109.
          debugfs_create_file("check_leaked_fds", 0664, idev->debug_root, idev,
110.
                      &debug_leak_fops);
111.
           return idev;
112.
113.
114.
       msm ion allocate:
       static void msm_ion_allocate(struct ion_platform_heap *heap)
115.
116.
117.
118.
          if (!heap->base && heap->extra_data) {
119.
              unsigned int align = 0;
120.
              switch (heap->type) {
121.
              /*获取align参数*/
122.
              case ION_HEAP_TYPE_CARVEOUT:
123.
                  align =
                  ((struct ion_co_heap_pdata *) heap->extra_data)->align;
124.
125.
                  break:
126.
              /*此type我们没使用到。*/
127.
              case ION_HEAP_TYPE_CP:
128.
                  struct ion_cp_heap_pdata *data =
129.
130.
                      (struct ion_cp_heap_pdata *)
131.
                      heap->extra data:
132.
                  if (data->reusable) {
133.
                      const struct fmem_data *fmem_info =
134.
                         fmem_get_info();
135.
                      heap->base = fmem info->phys;
                      data->virt addr = fmem info->virt:
136.
137.
                      pr_info("ION heap %s using FMEM\n", heap->name);
138.
                  } else if (data->mem_is_fmem) {
139.
                      const struct fmem_data *fmem_info =
140.
                          fmem_get_info();
                      heap->base = fmem_info->phys + fmem_info->size;
141.
142.
143.
                  align = data->align;
144.
                  break;
145.
146.
              default:
147.
                  break;
148.
149.
              if (align && !heap->base) {
                  /*获取heap的base address。*/
150.
151.
                  heap->base = msm_ion_get_base(heap->size,
152.
                                    heap->memory_type,
153.
                                    align);
                  if (!heap->base)
154.
                      pr_err("%s: could not get memory for heap %s "
155.
156.
                         "(id %x)\n", __func__, heap->name, heap->id);
157.
158.
159.
      }
160.
                                                                                                 关闭
161.
       static unsigned long msm_ion_get_base(unsigned long 
162.
                          unsigned int align)
163.
164.
           switch (memory_type) {
           /*我们定义的是ebi type,看见没,此函数在mem pool中分析过了。
165.
       原理就是使用Mempool 来管理分配内存。*/
166.
167.
           case ION EBI TYPE:
168.
              return allocate_contiguous_ebi_nomap(size, align);
169.
170.
          case ION_SMI_TYPE:
171.
              return allocate contiguous memory nomap(size, MEMTYPE SMI,
172.
                                  align):
173.
              break:
           default:
174.
              pr_err("%s: Unknown memory type %d\n", __func__, memory_type);
```

第13页 共69页 2017年09月07日 04:09

```
176.
               return 0;
177.
178.
179.
      ion heap create:
      struct ion_heap *ion_heap_create(struct ion_platform_heap *heap_data)
180.
181.
182.
           struct ion_heap *heap = NULL;
183.
           /*根据Heap type调用相应的创建函数。*/
184.
           switch (heap data->type) {
185.
          case ION_HEAP_TYPE_SYSTEM_CONTIG:
186.
              heap = ion_system_contig_heap_create(heap_data);
187.
              break:
188.
          case ION_HEAP_TYPE_SYSTEM:
189.
              heap = ion_system_heap_create(heap_data);
190.
              break;
          case ION_HEAP_TYPE_CARVEOUT:
191.
192.
             heap = ion_carveout_heap_create(heap_data);
193.
              break:
194.
          case ION_HEAP_TYPE_IOMMU:
195.
             heap = ion_iommu_heap_create(heap_data);
196.
              break;
          case ION_HEAP_TYPE_CP:
197.
198.
              heap = ion_cp_heap_create(heap_data);
199.
              break:
200.
      #ifdef CONFIG CMA
201.
          case ION_HEAP_TYPE_DMA:
202.
              heap = ion_cma_heap_create(heap_data);
203.
      #endif
204.
205
           default:
206.
             pr_err("%s: Invalid heap type %d\n", __func__,
207.
                     heap_data->type);
208.
              return ERR_PTR(-EINVAL);
209.
210.
211.
          if (IS_ERR_OR_NULL(heap)) {
212.
              pr_err("%s: error creating heap %s type %d base %lu size %u\n",
213.
                       _func__, heap_data->name, heap_data->type,
214.
                     heap_data->base, heap_data->size);
215.
              return ERR PTR(-EINVAL);
216.
           /*保存Heap的name,id和私有数据。*/
217.
218.
           heap->id = heap_data->id;
219.
220.
           heap->priv = heap_data->priv;
221.
           return heap;
222. }
 01.
       <span xmlns="http://www.w3.org</pre>
       /1999/xhtml" style="">static struct platform_driver msm_ion_driver = {
          .probe = msm_ion_probe,
 02
 03.
           .remove = msm_ion_remove,
           .driver = { .name = "ion-msm" }
 04.
 05.
      };
 06.
      static int __init msm_ion_init(void)
 07.
 08.
           /*调用msm_ion_probe */
           return platform_driver_register(&msm_ion_driver);
 10.
 11.
 12.
       static int msm_ion_probe(struct platform_device *pdev)
 13.
           /*即board-qrd7627a.c中的ion_pdata */
 15.
           struct ion_platform_data *pdata = pdev->dev.plat1
                                                                                                  关闭
           int err;
 16.
 17.
           int i:
 18.
 19.
           /*heap数量*/
 20.
           num_heaps = pdata->nr;
           /*分配struct ion_heap */
 22.
          heaps = kcalloc(pdata->nr, sizeof(struct ion_heap *), GFP_KERNEL);
 23.
 24.
           if (!heaps) {
 25.
               err = -ENOMEM;
 27.
 28.
           /*创建节点,最终是/dev/ion,供用户空间操作。*/
 29.
           idev = ion_device_create(NULL);
```

第14页 共69页 2017年09月07日 04:09

```
30.
          if (IS_ERR_OR_NULL(idev)) {
 31.
              err = PTR_ERR(idev);
 32.
              goto freeheaps;
 33.
          ·
/*最终是根据adjacent_mem_id 是否定义了来分配相邻内存,
 34.
      我们没用到,忽略此函数。*/
 35.
 36.
          msm_ion_heap_fixup(pdata->heaps, num_heaps);
 37.
 38.
          /^{*} create the heaps as specified in the board file ^{*}/
 39.
          for (i = 0; i < num_heaps; i++) {</pre>
 40.
              struct ion_platform_heap *heap_data = &pdata->heaps[i];
 41.
              /*分配ion*/
 42.
              msm_ion_allocate(heap_data);
 43.
              heap_data->has_outer_cache = pdata->has_outer_cache;
 45.
              /*创建ion heap。*/
              heaps[i] = ion_heap_create(heap_data);
 46.
 47.
              if (IS_ERR_OR_NULL(heaps[i])) {
 48.
                  heaps[i] = 0;
 49.
                  continue;
 50.
              } else {
                  if (heap_data->size)
 51.
                      pr_info("ION heap %s created at %lx "
 52.
                           "with size %x\n", heap_data->name,
 53.
 54.
                                    heap_data->base,
 55.
                                    heap_data->size);
 56.
 57.
                      pr info("ION heap %s created\n",
 58.
                                    heap_data->name);
 59.
 60.
              /*创建的heap添加到idev中,以便后续使用。*/
 61.
              ion_device_add_heap(idev, heaps[i]);
 62.
          /*检查heap之间是否有重叠部分*/
 63.
          check for heap overlap(pdata->heaps, num heaps);
 64.
 65.
          platform_set_drvdata(pdev, idev);
 66.
          return 0;
 67.
 68.
      freeheaps:
 69.
          kfree(heaps);
 70.
      out:
 71.
          return err;
 72.
 73.
       通过ion_device_create创建/dev/ion节点:
 74.
      struct ion_device *ion_device_create(long (*custom_ioctl)
 75.
 76.
                            (struct ion client *client.
 77.
                            unsigned int cmd.
 78.
                            unsigned long arg))
 79.
 80.
          struct ion_device *idev;
 81.
          int ret;
 82.
 83.
          idev = kzalloc(sizeof(struct ion_device), GFP_KERNEL);
 84.
          if (!idev)
 85.
              return ERR_PTR(-ENOMEM);
 86.
           /*是个misc设备*/
 87.
          idev->dev.minor = MISC DYNAMIC MINOR;
 88.
          /*节点名字为ion*/
          idev->dev.name = "ion";
 89.
 90.
          /*fops为ion_fops,所以对应ion的操作都会调用ion_fops的函数指针。*/
 91.
          idev->dev.fops = &ion_fops;
          idev->dev.parent = NULL;
 93.
          ret = misc register(&idev->dev);
 94.
          if (ret) {
                                                                                                 关闭
              pr_err("ion: failed to register misc device.")
 95
 96.
              return ERR_PTR(ret);
 98.
          /*创建debugfs目录,路径为/sys/kernel/debug/ion/*/
          idev->debug_root = debugfs_create_dir("ion", NULL);
 99.
          if (IS ERR OR NULL(idev->debug root))
100.
101.
              pr\_err("ion: failed to create debug files.\n");
102.
103.
          idev->custom_ioctl = custom_ioctl;
104.
          idev->buffers = RB_R00T;
105.
          mutex init(&idev->lock);
106.
          idev->heans = RB ROOT:
107.
          idev->clients = RB ROOT:
108.
          /*在ion目录下创建一个check_leaked_fds文件,用来检查ion的使用是否有内存泄漏。如果申请了ion之后
      不需要使用却没有释放,就会导致memory leak.*/
```

第15页 共69页 2017年09月07日 04:09

```
109.
           debugfs_create_file("check_leaked_fds", 0664, idev->debug_root, idev,
110.
                      &debug_leak_fops);
111.
112.
113.
114.
       msm_ion_allocate:
115.
       static void msm_ion_allocate(struct ion_platform_heap *heap)
116.
117.
118.
          if (!heap->base && heap->extra_data) {
119.
              unsigned int align = 0:
120.
              switch (heap->type) {
121.
              /*获取align参数*/
122.
              case ION_HEAP_TYPE_CARVEOUT:
123.
124.
                  ((struct ion_co_heap_pdata *) heap->extra_data)->align;
125.
                  break:
126.
              /*此type我们没使用到。*/
127.
              case ION_HEAP_TYPE_CP:
128.
129.
                  struct ion_cp_heap_pdata *data =
130.
                      (struct ion_cp_heap_pdata *)
131.
                      heap->extra data:
132.
                  if (data->reusable) {
133.
                      const struct fmem_data *fmem_info =
134.
                         fmem_get_info();
                      heap->base = fmem_info->phys;
135.
                      data->virt_addr = fmem_info->virt;
136.
137.
                      pr_info("ION heap %s using FMEM\n", heap->name);
138.
                  } else if (data->mem_is_fmem) {
139.
                      const struct fmem_data *fmem_info =
140.
                          fmem_get_info();
141.
                      heap->base = fmem_info->phys + fmem_info->size;
142.
                  align = data->align:
143.
144.
                  break:
145.
146.
              default:
                  break;
148.
149.
              if (align && !heap->base) {
150.
                  /*获取heap的base address。*/
151.
                  heap->base = msm_ion_get_base(heap->size,
                                    heap->memory_type,
152.
153.
                                    align);
                  if (!heap->base)
154.
155.
                      pr_err("%s: could not get memory for heap %s "
156.
                         "(id %x)\n", __func__, heap->name, heap->id);
157.
158.
159.
160.
      static unsigned long msm_ion_get_base(unsigned long size, int memory_type,
161.
162.
                          unsigned int align)
163.
164.
           switch (memory_type) {
           /*我们定义的是ebi type,看见没,此函数在mem pool中分析过了。
165.
      原理就是使用Mempool 来管理分配内存。*/
166.
          case ION EBI TYPE:
167.
168.
              return allocate_contiguous_ebi_nomap(size, align);
169.
              break:
170.
          case ION_SMI_TYPE:
171.
             return allocate_contiguous_memory_nomap(size, MEMTYPE_SMI,
172.
                                  align);
173.
              break:
                                                                                                 关闭
          default:
174.
175.
              pr_err("%s: Unknown memory type %d\n", __func__, memory_type);
176.
177.
178.
179.
       ion heap create:
180.
       struct ion_heap *ion_heap_create(struct ion_platform_heap *heap_data)
181.
182.
           struct ion_heap *heap = NULL;
183.
          /*根据Heap type调用相应的创建函数。*/
184.
          switch (heap data->type) {
185.
          case ION HEAP TYPE SYSTEM CONTIG:
186.
              heap = ion_system_contig_heap_create(heap_data);
              break;
187.
          case ION_HEAP_TYPE_SYSTEM:
```

第16页 共69页 2017年09月07日 04:09

```
189.
               heap = ion_system_heap_create(heap_data);
 190.
               break;
 191.
           case ION_HEAP_TYPE_CARVEOUT:
 192.
               heap = ion carveout heap create(heap data);
 193.
              break:
 194.
           case ION_HEAP_TYPE_IOMMU:
 195.
              heap = ion_iommu_heap_create(heap_data);
 196.
 197.
          case ION_HEAP_TYPE_CP:
 198.
             heap = ion_cp_heap_create(heap_data);
 199.
               break:
 200.
       #ifdef CONFIG CMA
 201.
           case ION_HEAP_TYPE_DMA:
               heap = ion_cma_heap_create(heap_data);
 202.
 203.
 204.
       #endif
 205.
           default:
 206
               pr_err("%s: Invalid heap type %d\n", __func__,
 207.
                     heap_data->type);
 208.
               return ERR_PTR(-EINVAL);
 209.
           }
 210.
 211.
           if (IS ERR OR NULL(heap)) {
 212.
               pr_err("%s: error creating heap %s type %d base %lu size %u\n",
 213.
                       _func__, heap_data->name, heap_data->type,
 214.
                      heap_data->base, heap_data->size);
 215.
               return ERR_PTR(-EINVAL);
 216.
           /*保存Heap的name,id和私有数据。*/
 217.
 218.
           heap->name = heap_data->name;
 219.
           heap->id = heap_data->id;
 220.
           heap->priv = heap_data->priv;
 221.
           return heap;
 222. }</span>
从下面的代码可以得知,ION_HEAP_TYPE_SYSTEM_CONTIG使用kmalloc创建
的, ION HEAP TYPE SYSTEM使用的是vmalloc, mion carveout heap create就是系统预分配了一片内存区域
供其使用。Ion在申请使用的时候,会根据当前的type来操作各自的heap->ops。分别看下三个函数:
  01.
       struct ion_heap *ion_system_contig_heap_create(struct ion_platform_heap *pheap)
  02.
           struct ion_heap *heap;
  04.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
  05.
  06.
           if (!heap)
  07.
               return ERR_PTR(-ENOMEM);
           /*使用的是kmalloc_ops,上篇有提到哦*/
  09.
           heap->ops = &kmalloc_ops;
           heap->type = ION_HEAP_TYPE_SYSTEM_CONTIG;
  10.
  11.
           system_heap_contig_has_outer_cache = pheap->has_outer_cache;
  12.
           return heap;
  13.
  14.
       struct ion_heap *ion_system_heap_create(struct ion_platform_heap *pheap)
  15.
  16.
           struct ion_heap *heap;
  17.
  18.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
  19.
           if (!heap)
  20.
               return ERR_PTR(-ENOMEM);
           /*和上面函数的区别仅在于ops*/
  21.
           heap->ops = &vmalloc_ops;
  23.
           heap->type = ION_HEAP_TYPE_SYSTEM;
                                                                                                关闭
  24.
           system_heap_has_outer_cache = pheap->has_outer_cache....,
  25.
           return heap;
  26.
  27.
       struct ion_heap *ion_carveout_heap_create(struct ion_platform_heap *heap_data)
  28.
  29.
           struct ion carveout heap *carveout heap;
  30.
           int ret:
  31.
           carveout_heap = kzalloc(sizeof(struct ion_carveout_heap), GFP_KERNEL);
  32.
  33.
           if (!carveout_heap)
  34.
               return ERR_PTR(-ENOMEM);
           /* 重新创建一个新的pool,这里有点想不通的是为什么不直接使用全局的mempools呢?*/
  35.
  36.
           carveout_heap->pool = gen_pool_create(12, -1);
  37.
           if (!carveout_heap->pool) {
```

第17页 共69页 2017年09月07日 04:09

```
38.
               kfree(carveout_heap);
 39.
               return ERR_PTR(-ENOMEM);
 40.
 41.
           carveout heap->base = heap data->base:
 42.
           ret = gen_pool_add(carveout_heap->pool, carveout_heap->base,
 43.
                   heap_data->size, -1);
           if (ret < 0) {
 44.
 45.
              gen_pool_destroy(carveout_heap->pool);
 46.
               kfree(carveout_heap);
 47.
              return ERR_PTR(-EINVAL);
 48.
 49.
           carveout_heap->heap.ops = &carveout_heap_ops;
 50.
           carveout_heap->heap.type = ION_HEAP_TYPE_CARVEOUT;
           carveout_heap->allocated_bytes = 0;
 51.
 52.
           carveout_heap->total_size = heap_data->size;
 53.
           carveout_heap->has_outer_cache = heap_data->has_outer_cache;
 54.
 55.
           if (heap_data->extra_data) {
               struct ion_co_heap_pdata *extra_data =
 56.
                      heap_data->extra_data;
 57.
 58.
              if (extra_data->setup_region)
 59.
 60.
                   carveout_heap->bus_id = extra_data->setup_region();
 61.
               if (extra_data->request_region)
 62.
                   carveout_heap->request_region =
 63.
                           extra_data->request_region;
               if (extra_data->release_region)
 65.
                   carveout heap->release region =
 66.
                           extra_data->release_region;
 67.
 68.
           return &carveout_heap->heap;
 69.
 70.
       Heap创建完成,然后保存到idev中:
 71.
 72.
       void ion_device_add_heap(struct ion_device *dev, struct ion_heap *heap)
 73.
 74.
           struct rb_node **p = &dev->heaps.rb_node;
 75.
           struct rb_node *parent = NULL;
 76.
           struct ion_heap *entry;
 77.
           if (!heap->ops->allocate || !heap->ops->free || !heap->ops->map_dma ||
 78.
 79.
               !heap->ops->unmap_dma)
 80.
               pr\_err("%s: can not add heap with invalid ops struct.\n",
 81.
 82.
           heap->dev = dev;
 83.
 84.
           mutex_lock(&dev->lock);
 85.
           while (*p) {
 86.
              parent = *p;
 87.
               entry = rb_entry(parent, struct ion_heap, node);
 88.
 89.
              if (heap->id < entry->id) {
 90.
                   p = &(*p)->rb_left;
 91.
              } else if (heap->id > entry->id ) {
 92.
                   p = &(*p) -> rb_right;
 93.
              } else {
                   pr_err("%s: can not insert multiple heaps with "
                      "id %d\n", __func__, heap->id);
 95.
 96.
                   goto end:
 97.
              }
 98.
 99.
           /*使用红黑树保存*/
100.
           rb_link_node(&heap->node, parent, p);
101.
           rb insert color(&heap->node, &dev->heaps);
           /*以heap name创建fs,位于ion目录下。如vamlloc, camera nreview audio 等*/
102.
103.
           debugfs_create_file(heap->name, 0664, dev->debug_
104.
                       &debug_heap_fops);
105.
106.
           mutex_unlock(&dev->lock);
107.
      }
       <span xmlns="http://www.w3.org</pre>
 01.
       /1999/xhtml" style="">struct ion_heap *ion_system_contig_heap_create(struct ion_platforr
 02.
 03.
           struct ion_heap *heap;
 04.
 05.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
 06.
           if (!heap)
```

第18页 共69页 2017年09月07日 04:09

```
07.
             return ERR_PTR(-ENOMEM);
08.
          /*使用的是kmalloc_ops,上篇有提到哦*/
09.
          heap->ops = &kmalloc_ops;
          heap->type = ION_HEAP_TYPE_SYSTEM_CONTIG;
10.
11.
          system_heap_contig_has_outer_cache = pheap->has_outer_cache;
12.
13.
     struct ion_heap *ion_system_heap_create(struct ion_platform_heap *pheap)
14.
15.
16.
          struct ion_heap *heap;
17.
18.
          heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
19.
          if (!heap)
20.
              return ERR_PTR(-ENOMEM);
          /*和上面函数的区别仅在于ops*/
22.
         heap->ops = &vmalloc ops:
         heap->type = ION_HEAP_TYPE_SYSTEM;
23.
24.
          system_heap_has_outer_cache = pheap->has_outer_cache;
25.
          return heap;
26.
27.
      struct ion_heap *ion_carveout_heap_create(struct ion_platform_heap *heap_data)
28.
29.
          struct ion_carveout_heap *carveout_heap;
30.
31.
32.
          carveout_heap = kzalloc(sizeof(struct ion_carveout_heap), GFP_KERNEL);
33.
          if (!carveout_heap)
             return ERR PTR(-ENOMEM);
34.
          - * 重新创建一个新的pool,这里有点想不通的是为什么不直接使用全局的mempools呢?*/
35.
36.
          carveout_heap->pool = gen_pool_create(12, -1);
37.
          if (!carveout_heap->pool) {
38.
              kfree(carveout_heap);
39.
             return ERR_PTR(-ENOMEM);
40.
          carveout heap->base = heap data->base:
41.
42.
          ret = gen_pool_add(carveout_heap->pool, carveout_heap->base,
43.
                  heap_data->size, -1);
44.
          if (ret < 0) {
             gen_pool_destroy(carveout_heap->pool);
46.
             kfree(carveout heap);
47.
             return ERR_PTR(-EINVAL);
48.
49.
          carveout_heap->heap.ops = &carveout_heap_ops;
          carveout_heap->heap.type = ION_HEAP_TYPE_CARVEOUT;
50.
51.
          carveout_heap->allocated_bytes = 0;
52.
          carveout heap->total size = heap data->size;
53.
          carveout_heap->has_outer_cache = heap_data->has_outer_cache;
54.
55.
          if (heap_data->extra_data) {
56.
             struct ion_co_heap_pdata *extra_data =
                     heap_data->extra_data;
58.
             if (extra_data->setup_region)
59.
60.
                  carveout_heap->bus_id = extra_data->setup_region();
61.
             if (extra_data->request_region)
                  carveout_heap->request_region =
63.
                          extra_data->request_region;
64.
             if (extra data->release region)
65.
                  carveout heap->release region =
66.
                         extra_data->release_region;
67.
68.
          return &carveout_heap->heap;
69.
70.
      Hean创建完成,然后保存到idev中:
71.
                                                                                                  关闭
72.
      void ion_device_add_heap(struct ion_device *dev, struct)
73.
          struct rb_node **p = &dev->heaps.rb_node;
74.
75.
          struct rb_node *parent = NULL;
          struct ion heap *entry;
76.
77.
78.
          if (!heap->ops->allocate || !heap->ops->free || !heap->ops->map_dma ||
79.
              !heap->ops->unmap_dma)
80.
             pr\_err("%s: can not add heap with invalid ops struct.\n",
81.
                     __func__);
82.
         heap->dev = dev:
83.
84.
          mutex_lock(&dev->lock);
85.
          while (*p) {
             parent = *p;
```

第19页 共69页 2017年09月07日 04:09

```
87.
              entry = rb_entry(parent, struct ion_heap, node);
 88.
              if (heap->id < entry->id) {
 90.
                  p = &(*p) -> rb left:
              } else if (heap->id > entry->id ) {
 91.
 92.
                  p = &(*p) -> rb_right;
 93.
              } else {
                 pr_err("%s: can not insert multiple heaps with "
                      "id %d\n", __func__, heap->id);
                  qoto end:
              }
 97.
 98.
           /*使用红黑树保存*/
 aa
           rb_link_node(&heap->node, parent, p);
100.
101.
           rb_insert_color(&heap->node, &dev->heaps);
102.
           /*以heap name创建fs,位于ion目录下。如vamlloc, camera_preview , audio 等*/
103.
          debugfs_create_file(heap->name, 0664, dev->debug_root, heap,
104
                      &debug_heap_fops);
105.
106.
          mutex_unlock(&dev->lock);
107.
108. </span>
```

到此,ION初始化已经完成了。接下来该如何使用呢?嗯,通过前面创建的misc设备也就是idev了! 有个fops为ion fops吗?先来看下用户空间如何使用ION,最后看内核空间如何使用。

## ION用户空间使用

```
01.
      Ion_fops结构如下:
02.
      static const struct file_operations ion_fops = {
         .owner = THIS_MODULE,
04.
          .open
                         = ion open,
05.
          .release
                         = ion release.
          .unlocked_ioctl = ion_ioctl,
06.
07.
08.
      用户空间都是通过ioctl来控制。先看ion_open.
09.
10.
11.
      static int ion open(struct inode *inode, struct file *file)
12.
13.
          struct miscdevice *miscdev = file->private_data;
14.
          struct ion_device *dev = container_of(miscdev, struct ion_device, dev);
          struct ion_client *client;
16.
          char debug_name[64];
17.
         pr_debug("%s: %d\n", __func__, __LINE__);
snprintf(debug_name, 64, "%u", task_pid_nr(current->group_leader));
18.
19.
          /*根据idev和task pid为name创建ion client*/
20.
21.
          client = ion_client_create(dev, -1, debug_name);
22.
         if (IS_ERR_OR_NULL(client))
23.
             return PTR ERR(client):
24.
          file->private_data = client;
25.
26.
          return 0;
27. }
01.
      <span xmlns="http://www.w3.org/1999/xhtml" style="">Ion_fops结构如下:
      static const struct file operations ion fops = {
02.
03.
         .owner
                        = THIS MODULE,
          . open
04.
                         = ion_open,
05.
                         = ion_release,
                                                                                                   关闭
          .unlocked_ioctl = ion_ioctl,
07.
08.
      用户空间都是通过ioctl来控制。先看ion open.
09.
10.
11.
      static int ion_open(struct inode *inode, struct file *file)
12.
          struct miscdevice *miscdev = file->private_data;
13.
14.
          struct ion_device *dev = container_of(miscdev, struct ion_device, dev);
          struct ion_client *client;
15.
16.
          char debug_name[64];
17.
          pr_debug("%s: %d\n", __func__, __LINE__);
18.
          snprintf(debug_name, 64, "%u", task_pid_nr(current->group_leader));
19.
```

第20页 共69页 2017年09月07日 04:09

```
20.
           /*根据idev和task pid为name创建ion client*/
 21.
           client = ion_client_create(dev, -1, debug_name);
           if (IS_ERR_OR_NULL(client))
 23.
               return PTR_ERR(client);
 24.
           file->private_data = client;
 25.
 26.
           return 0;
 27. }</span>
前一篇文章有说到,要使用ION,必须要先创建ion client,因此用户空间在open ion的时候创建了client.
                   CP
       [cpp]
       struct ion_client *ion_client_create(struct ion_device *dev,
 01.
 02.
                            unsigned int heap mask,
 03.
                            const char *name)
  04
 05.
           struct ion_client *client;
 06.
           struct task_struct *task;
 07.
           struct rb node **p;
           struct rb_node *parent = NULL;
 08.
 09.
           struct ion_client *entry;
 10.
           pid_t pid;
           unsigned int name_len;
 12.
 13.
           if (!name) {
               pr_err("%s: Name cannot be null\n", __func__);
 14.
 15.
               return ERR_PTR(-EINVAL);
 16.
 17.
           name_len = strnlen(name, 64);
 18.
 19.
           get task struct(current->group leader);
 20.
           task lock(current->group leader);
 21.
           pid = task_pid_nr(current->group_leader);
 22.
           /* don't bother to store task struct for kernel threads,
 23.
              they can't be killed anyway */
 24.
           if (current->group_leader->flags & PF_KTHREAD) {
 25.
               put task struct(current->group leader);
 26.
               task = NULL;
 27.
           } else {
 28.
               task = current->group_leader;
 29.
 30.
           task_unlock(current->group_leader);
 31.
           /*分配ion client struct.*,
           client = kzalloc(sizeof(struct ion client), GFP KERNEL);
 32.
 33.
           if (!client) {
 34.
               if (task)
 35.
                   put_task_struct(current->group_leader);
 36.
               return ERR_PTR(-ENOMEM);
 37.
           /*下面就是保存一系列参数了。*/
 38.
 39.
           client->dev = dev;
  40.
           client->handles = RB_ROOT;
  41.
           mutex_init(&client->lock);
 42.
           client->name = kzalloc(name_len+1, GFP_KERNEL);
 43.
           if (!client->name) {
 44.
 45.
               put_task_struct(current->group_leader);
  46.
               kfree(client);
               return ERR_PTR(-ENOMEM);
  47.
  48.
           } else {
 49.
               strlcpy(client->name, name, name_len+1);
 50.
 51.
  52.
           client->heap_mask = heap_mask;
                                                                                                    关闭
  53.
           client->task = task;
           client->pid = pid;
 55.
 56.
           mutex lock(&dev->lock);
 57.
           p = &dev->clients.rb_node;
 58.
           while (*p) {
 59.
               parent = *p;
  60.
               entry = rb_entry(parent, struct ion_client, node);
 61.
 62.
               if (client < entry)</pre>
 63.
                   p = &(*p)->rb_left;
  64.
                else if (client > entry)
  65.
                   p = &(*p)->rb_right;
```

第21页 共69页 2017年09月07日 04:09

```
67.
          /*当前client添加到idev的clients根树上去。*/
68.
          rb_link_node(&client->node, parent, p);
69.
          rb_insert_color(&client->node, &dev->clients);
70.
          /*在ION先创建的文件名字是以pid命名的。*/
71.
72.
          client->debug_root = debugfs_create_file(name, 0664,
73.
                               dev->debug_root, client,
74.
                               &debug_client_fops);
75.
         mutex_unlock(&dev->lock);
76.
77.
         return client:
78. }
01.
      <span xmlns="http://www.w3.org</pre>
      /1999/xhtml" style="">struct ion_client *ion_client_create(struct ion_devic
02.
                           unsigned int heap_mask,
03.
                           const char *name)
04.
          struct ion_client *client;
05.
06.
          struct task_struct *task;
07.
          struct rb_node **p;
          struct rb_node *parent = NULL;
08.
09.
          struct ion client *entry:
          pid_t pid;
10.
11.
          unsigned int name_len;
12.
13.
          if (!name) {
14.
             pr_err("%s: Name cannot be null\n", __func__);
              return ERR PTR(-EINVAL);
15.
16.
17.
         name_len = strnlen(name, 64);
18.
19.
          get_task_struct(current->group_leader);
          task_lock(current->group_leader);
21.
          pid = task pid nr(current->group leader);
22.
          /* don't bother to store task struct for kernel threads,
23.
            they can't be killed anyway */
24.
          if (current->group_leader->flags & PF_KTHREAD) {
25.
             put_task_struct(current->group_leader);
26.
              task = NULL;
27.
         } else {
             task = current->group leader;
28.
29.
30.
          task_unlock(current->group_leader);
31.
          /*分配ion client struct.*/
          client = kzalloc(sizeof(struct ion_client), GFP_KERNEL);
33.
         if (!client) {
34.
             if (task)
35.
                  put_task_struct(current->group_leader);
36.
              return ERR_PTR(-ENOMEM);
37.
          /*下面就是保存一系列参数了。*/
         client->dev = dev;
39.
          client->handles = RB ROOT;
40.
41.
          mutex_init(&client->lock);
42.
43.
          client->name = kzalloc(name_len+1, GFP_KERNEL);
          if (!client->name) {
45.
              put_task_struct(current->group_leader);
46.
              kfree(client);
47.
             return ERR_PTR(-ENOMEM);
48.
         } else {
49.
              strlcpy(client->name, name, name_len+1);
50.
                                                                                                   关闭
51.
52.
          client->heap_mask = heap_mask;
53.
          client->task = task;
54.
          client->pid = pid;
          mutex_lock(&dev->lock);
57.
         p = &dev->clients.rb_node;
58.
         while (*p) {
             parent = *p;
59.
60.
              entry = rb_entry(parent, struct ion_client, node);
62.
              if (client < entry)</pre>
63.
                 p = &(*p)->rb_left;
64.
              else if (client > entry)
```

第22页 共69页 2017年09月07日 04:09

```
65.
                   p = &(*p)->rb_right;
  66.
           /*当前client添加到idev的clients根树上去。*/
  67.
           rb_link_node(&client->node, parent, p);
  68.
           rb insert color(&client->node, &dev->clients):
  69.
  70.
  71.
           /*在ION先创建的文件名字是以pid命名的。*/
  72.
           client->debug_root = debugfs_create_file(name, 0664,
  73.
                                dev->debug_root, client,
                               &debug_client_fops);
           mutex unlock(&dev->lock):
  75.
  76.
  77.
           return client;
  78.
有了client之后,用户程序就可以开始申请分配ION buffer了!通过ioctl命令实现。
ion_ioct函数有若干个cmd, ION_IOC_ALLOC和ION_IOC_FREE相对应,表示申请和释放buffer。
使用前先要调用ION_IOC_MAP才能得到buffer address,而ION_IOC_IMPORT是为了将这块内存
间另一个进程。
                   CP
       [cpp]
  01.
       static long ion_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
  02.
           struct ion_client *client = filp->private_data;
  03.
  04.
  05.
           switch (cmd) {
  06.
           case ION_IOC_ALLOC:
  08.
               struct ion_allocation_data data;
  09.
               if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
  10.
                   return -EFAULT;
  11.
  12.
               /*分配buffer.*/
               data.handle = ion_alloc(client, data.len, data.align,
  13.
  14.
                                data.flags);
  15.
  16.
               if (IS ERR(data.handle))
  17.
                   return PTR_ERR(data.handle);
  18.
               if (copy_to_user((void __user *)arg, &data, sizeof(data))) {
  20.
                   ion_free(client, data.handle);
                   return -EFAULT;
  21.
  22.
  23.
               break:
  24.
  25.
           case ION_IOC_FREE:
  26.
  27.
               struct ion handle data data;
  28.
               bool valid:
  29.
  30.
               if (copy_from_user(&data, (void __user *)arg,
  31.
                         sizeof(struct ion_handle_data)))
  32.
                   return -EFAULT;
               mutex_lock(&client->lock);
  33.
               valid = ion_handle_validate(client, data.handle);
  34.
  35.
               mutex_unlock(&client->lock);
  36.
               if (!valid)
  37.
                   return -EINVAL;
  38.
               ion_free(client, data.handle);
  39.
               break;
  40.
           case ION IOC MAP:
  41.
  42.
           case ION_IOC_SHARE:
                                                                                                 关闭
  43.
  44.
               struct ion_fd_data data;
  45.
               int ret:
               if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
  46.
  47.
                   return -EFAULT;
  48.
               /*判断当前cmd是否被调用过了,调用过就返回,否则设置flags.*/
  49.
               ret = ion_share_set_flags(client, data.handle, filp->f_flags);
  50.
               if (ret)
  51.
                   return ret:
  52.
  53.
               data.fd = ion_share_dma_buf(client, data.handle);
  54.
               if (copy_to_user((void __user *)arg, &data, sizeof(data)))
                   return -EFAULT;
               if (data.fd < 0)</pre>
```

第23页 共69页 2017年09月07日 04:09

```
57.
                                       return data.fd;
58.
59.
                     case ION_IOC_IMPORT:
60.
61.
62.
                              struct ion_fd_data data;
63.
                              int ret = 0;
                             if (copy_from_user(&data, (void __user *)arg,
65.
                                                    sizeof(struct ion_fd_data)))
66.
                                     return -EFAULT;
67.
                             data.handle = ion_import_dma_buf(client, data.fd);
68.
                             if (IS_ERR(data.handle))
69.
                                       data.handle = NULL;
70.
                              if (copy_to_user((void __user *)arg, &data,
                                               sizeof(struct ion_fd_data)))
72.
                                      return -EFAULT;
                             if (ret < 0)
73.
74.
                                      return ret;
75.
                             break;
76.
77.
                     case ION_IOC_CUSTOM:
78.
             ~~snip
79.
                     case ION_IOC_CLEAN_CACHES:
80.
                     case ION_IOC_INV_CACHES:
81.
                     case ION_IOC_CLEAN_INV_CACHES:
82.
                      ~~snip
83.
                     case ION_IOC_GET_FLAGS:
84.
            ~~snip
                     default:
85.
86.
                            return -ENOTTY;
87.
88.
89.
             <span xmlns="http://www.w3.org</pre>
01.
             /1999/xhtml" style="">static long ion_ioctl(struct file *filp, unsigned int cmd, uns
02.
03.
                      struct ion_client *client = filp->private_data;
04.
05.
                      switch (cmd) {
06.
                     case ION IOC ALLOC:
07.
08.
                              struct ion allocation data data;
09.
10.
                              if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
                                       return -EFAULT;
12.
                              /*分配buffer.*/
                             data.handle = ion alloc(client, data.len, data.align,
13.
14.
                                                                   data.flags);
15.
16.
                             if (IS_ERR(data.handle))
17.
                                       return PTR_ERR(data.handle);
18.
                              if (copy_to_user((void __user *)arg, &data, sizeof(data))) {
19.
20.
                                       ion_free(client, data.handle);
21.
                                       return -EFAULT;
22.
24.
                     }
25.
                     case ION_IOC_FREE:
26.
27.
                              struct ion_handle_data data;
28.
29.
                                                                                                                                                                                                                   关闭
                              if (copy_from_user(&data, (void __user *)arg,
30.
31.
                                                   sizeof(struct ion_handle_data)))
32.
                                      return -EFAULT;
33.
                              mutex_lock(&client->lock);
34.
                              valid = ion_handle_validate(client, data.handle);
                              mutex_unlock(&client->lock);
36.
                              if (!valid)
                                      return -EINVAL:
37.
                              ion_free(client, data.handle);
38.
39.
                              break;
40.
41.
                      case ION_IOC_MAP:
42.
                     case ION_IOC_SHARE:
43.
                      {
```

第24页 共69页 2017年09月07日 04:09

关闭

```
44.
               struct ion_fd_data data;
  45.
               int ret;
  46.
               if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
  47.
                   return -EFAULT:
               /*判断当前cmd是否被调用过了,调用过就返回,否则设置flags.*/
  48.
  49.
               ret = ion_share_set_flags(client, data.handle, filp->f_flags);
  50.
               if (ret)
  51.
                   return ret;
  52.
  53.
               data.fd = ion_share_dma_buf(client, data.handle);
  54.
               if (copy_to_user((void __user *)arg, &data, sizeof(data)))
  55.
                   return -EFAULT:
  56.
               if (data.fd < 0)
  57.
                   return data.fd;
  59.
           case ION IOC IMPORT:
  60.
  61.
  62.
               struct ion_fd_data data;
  63.
               int ret = 0;
  64.
               if (copy_from_user(&data, (void __user *)arg,
  65.
                          sizeof(struct ion_fd_data)))
  66.
                   return -EFAULT:
  67.
               data.handle = ion_import_dma_buf(client, data.fd);
  68.
               if (IS_ERR(data.handle))
  69.
                   data.handle = NULL;
               if (copy_to_user((void __user *)arg, &data,
                        sizeof(struct ion_fd_data)))
  71.
                   return -EFAULT:
  72.
  73.
               if (ret < 0)
  74.
                   return ret;
  75.
               break;
  76.
  77.
           case ION IOC CUSTOM:
  78.
       ~~snip
  79.
           case ION_IOC_CLEAN_CACHES:
  80.
           case ION_IOC_INV_CACHES:
  81.
           case ION_IOC_CLEAN_INV_CACHES:
  82.
           ~~snip
  83.
           case ION IOC GET FLAGS:
       ~~snip
  84.
  85.
           default:
  86.
               return -ENOTTY;
  87.
  88.
           return 0;
  89. }</span>
下面分小节说明分配和共享的原理。
ION IOC ALLOC
  01.
       struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
  02.
                        size_t align, unsigned int flags)
  03.
  04.
  05.
  06.
           mutex_lock(&dev->lock);
  07.
           /*循环遍历当前Heap链表。*/
```

```
08.
          for (n = rb_first(&dev->heaps); n != NULL; n = rb_next(n)) {
             struct ion_heap *heap = rb_entry(n, struct ion_heap, node);
10.
     /*只有heap type和id都符合才去创建buffer.*/
             /* if the client doesn't support this heap type */
11.
12.
             if (!((1 << heap->type) & client->heap_mask);
13.
                 continue;
             /^{\star} if the caller didn't specify this heap type ^{\star}/
14.
15.
             if (!((1 << heap->id) & flags))
                 continue;
16.
17.
              /* Do not allow un-secure heap if secure is specified */
18.
             if (secure_allocation && (heap->type != ION_HEAP_TYPE_CP))
19.
20.
             buffer = ion_buffer_create(heap, dev, len, align, flags);
21.
22.
23.
         mutex unlock(&dev->lock);
24.
25.
         /*创建了buffer之后,就相应地创建handle来管理buffer.*/
```

第25页 共69页 2017年09月07日 04:09

```
27.
         handle = ion handle create(client, buffer);
28.
29.
       ~snip
30.
     }
31.
     找到Heap之后调用ion_buffer_create:
32.
33.
      static struct ion_buffer *ion_buffer_create(struct ion_heap *heap,
34.
                          struct ion_device *dev,
35.
                          unsigned long len,
36.
                          unsigned long align,
37.
                          unsigned long flags)
38.
39.
          struct ion_buffer *buffer;
40.
          struct sg_table *table;
          int ret;
42.
          /*分配struct ion buffer,用来管理buffer.*/
         buffer = kzalloc(sizeof(struct ion_buffer), GFP_KERNEL);
43.
44.
          if (!buffer)
45.
              return ERR_PTR(-ENOMEM);
46.
47.
          buffer->heap = heap;
          kref_init(&buffer->ref);
48.
          /*调用相应heap type的ops allocate。还记得前面有提到过不同种类的ops吗,
49.
50.
     如carveout_heap_ops , vmalloc_ops 。*/
51.
          ret = heap->ops->allocate(heap, buffer, len, align, flags);
52.
          if (ret) {
53.
             kfree(buffer);
             return ERR PTR(ret);
54.
55.
56.
57.
          buffer->dev = dev;
58.
          buffer->size = len;
59.
          /*http://lwn.net/Articles/263343/*/
60.
          table = buffer->heap->ops->map dma(buffer->heap, buffer);
61.
         if (IS_ERR_OR_NULL(table)) {
62.
             heap->ops->free(buffer);
63.
             kfree(buffer);
64.
             return ERR_PTR(PTR_ERR(table));
65.
66.
         buffer->sq table = table;
67.
68.
          mutex_init(&buffer->lock);
69
          /*将当前ion buffer添加到idev 的buffers 树上统一管理。*/
70.
          ion_buffer_add(dev, buffer);
71.
          return buffer;
72. }
     [cpp]
     <span xmlns="http://www.w3.org</pre>
01.
     /1999/xhtml" style="">struct ion_handle *ion_alloc(struct ion_client *client, size_t ler
02.
                      size_t align, unsigned int flags)
03.
04.
05.
06.
         mutex lock(&dev->lock);
07.
          /*循环遍历当前Heap链表。*/
08.
          for (n = rb_first(&dev->heaps); n != NULL; n = rb_next(n)) {
09.
              struct ion_heap *heap = rb_entry(n, struct ion_heap, node);
     /*只有heap type和id都符合才去创建buffer.*/
             /* if the client doesn't support this heap type */
11.
12.
             if (!((1 << heap->type) & client->heap_mask))
13.
                 continue:
             /^{\star} if the caller didn't specify this heap type ^{\star}/
14.
             if (!((1 << heap->id) & flags))
                                                                                                  关闭
              /* Do not allow un-secure heap if secure is specified */
17.
             if (secure_allocation && (heap->type != ION_HEAP_TYPE_CP))
18.
                 continue;
19.
20.
             buffer = ion_buffer_create(heap, dev, len, align, flags);
21.
     ~~snip
22.
23.
         mutex_unlock(&dev->lock);
24.
25.
26.
         /*创建了buffer之后,就相应地创建handle来管理buffer.*/
27.
          handle = ion_handle_create(client, buffer);
28.
29.
     ~~snip
30. }
```

第26页 共69页 2017年09月07日 04:09

```
31.
32.
      找到Heap之后调用ion_buffer_create:
33.
      static struct ion_buffer *ion_buffer_create(struct ion_heap *heap,
34.
                          struct ion device *dev.
                          unsigned long len,
35.
36.
                          unsigned long align,
37.
                          unsigned long flags)
38.
39.
          struct ion_buffer *buffer;
40.
          struct sg_table *table;
41.
         int ret:
42.
          /*分配struct ion buffer,用来管理buffer.*/
43.
         buffer = kzalloc(sizeof(struct ion_buffer), GFP_KERNEL);
44.
             return ERR_PTR(-ENOMEM);
46.
         buffer->heap = heap;
47.
48.
          kref_init(&buffer->ref);
49.
          /*调用相应heap type的ops allocate。还记得前面有提到过不同种类的ops吗,
      如carveout_heap_ops , vmalloc_ops 。*/
50.
51.
         ret = heap->ops->allocate(heap, buffer, len, align, flags);
52.
         if (ret) {
53.
             kfree(buffer);
54.
             return ERR_PTR(ret);
55.
56.
         buffer->dev = dev;
         buffer->size = len;
58.
         /*http://lwn.net/Articles/263343/*/
59.
60.
          table = buffer->heap->ops->map_dma(buffer->heap, buffer);
61.
         if (IS_ERR_OR_NULL(table)) {
             heap->ops->free(buffer);
63.
             kfree(buffer);
             return ERR_PTR(PTR_ERR(table));
64.
65.
66.
         buffer->sg_table = table;
67.
68.
          mutex_init(&buffer->lock);
          /*将当前ion buffer添加到idev 的buffers 树上统一管理。*/
70.
         ion_buffer_add(dev, buffer);
         return buffer:
71.
72.
     }</span>
01.
      static struct ion_handle *ion_handle_create(struct ion_client *client,
              struct ion_buffer *buffer)
02.
03.
04.
      struct ion_handle *handle;
05.
       /*分配struct ion_handle.*/
      handle = kzalloc(sizeof(struct ion_handle), GFP_KERNEL);
07.
      if (!handle)
       return ERR PTR(-ENOMEM):
08.
09.
      kref_init(&handle->ref);
10.
       rb_init_node(&handle->node);
11.
      handle->client = client; //client放入handle中
12.
      ion_buffer_get(buffer); //引用计数加1
      handle->buffer = buffer; //buffer也放入handle中 return handle;
13.
14.
     15.
01.
      <span xmlns="http://www.w3.org
      /1999/xhtml" style="">static struct ion_handle *ion_handle_create(struct ion_client *cl:
02.
              struct ion_buffer *buffer)
03.
04.
      struct ion handle *handle;
05.
      /*分配struct ion_handle.*/
06.
      handle = kzalloc(sizeof(struct ion_handle), GFP_KERNEL);
07.
      if (!handle)
       return ERR_PTR(-ENOMEM);
09.
      kref_init(&handle->ref);
      rb init node(&handle->node);
10.
11.
      handle->client = client; //client放入handle中
12.
      ion_buffer_get(buffer); //引用计数加1
      handle->buffer = buffer; //buffer也放入handle中</span><span xmlns="http:
      //www.w3.org/1999/xhtml" style=""> return handle;
15.
     </span>
```

第27页 共69页 2017年09月07日 04:09

```
先拿heap type为ION_HEAP_TYPE_CARVEOUT为例,看下它是如何分配buffer的。
allocate对应ion carveout heap allocate。
                   CP
  01.
       static int ion_carveout_heap_allocate(struct ion_heap *heap,
  02.
                             struct ion buffer *buffer.
                             unsigned long size, unsigned long align,
  03.
  04
                             unsigned long flags)
  05.
       {
           buffer->priv_phys = ion_carveout_allocate(heap, size, align);
  06.
  07.
           return buffer->priv_phys == ION_CARVEOUT_ALLOCATE_FAIL ? -ENOMEM : 0;
  08.
  09.
       ion phys addr t ion carveout allocate(struct ion heap *heap.
  10.
                             unsigned long size,
  11.
                             unsigned long align)
  12.
  13.
           struct ion_carveout_heap *carveout_heap =
               container of(heap, struct ion carveout heap, heap):
  14.
           /*通过创建的mem pool来管理buffer,由于这块buffer在初始化的
  15.
  16.
       时候就预留了,现在只要从上面拿一块区域就可以了。*/
  17.
           unsigned long offset = gen_pool_alloc_aligned(carveout_heap->pool,
  18.
                                   size, ilog2(align));
  19.
           /*分配不成功可能是没有内存空间可供分配了或者是有碎片导致的。*/
           if (!offset) {
  20.
  21.
               if ((carveout_heap->total size -
  22.
                     carveout_heap->allocated_bytes) >= size)
  23.
                   pr\_debug("%s: heap %s has enough memory (%lx) but"
  24.
                        " the allocation of size %lx still failed.'
  25.
                       " Memory is probably fragmented.",
                       func . heap->name.
  26.
  27.
                       carveout_heap->total_size -
  28.
                       carveout_heap->allocated_bytes, size);
  29.
               return ION_CARVEOUT_ALLOCATE_FAIL;
           -
/*已经分配掉的内存字节。*/
  31.
  32.
           carveout heap->allocated bytes += size;
  33.
           return offset;
  34.
  01.
        <span xmlns="http://www.w3.org</pre>
        /1999/xhtml" style="">static int ion_carveout_heap_allocate(struct ion_heap *heap,
  02.
                             struct ion_buffer *buffer,
  03.
                             unsigned long size, unsigned long align,
  04
                             unsigned long flags)
  05.
  06.
           buffer->priv_phys = ion_carveout_allocate(heap, size, align);
  07.
           return buffer->priv_phys == ION_CARVEOUT_ALLOCATE_FAIL ? -ENOMEM : 0;
  08.
  09.
       ion phys addr t ion carveout allocate(struct ion heap *heap.
  10.
                             unsigned long size.
  11.
                             unsigned long align)
  12.
           struct ion_carveout_heap *carveout_heap =
  13.
  14.
               container_of(heap, struct ion_carveout_heap, heap);
           /*通过创建的mem pool来管理buffer,由于这块buffer在初始化的
  15.
       时候就预留了,现在只要从上面拿一块区域就可以了。*/
  16.
  17.
           unsigned long offset = gen_pool_alloc_aligned(carveout_heap->pool,
  18.
                                   size, ilog2(align));
           /*分配不成功可能是没有内存空间可供分配了或者是有碎片导致的。*/
  19.
           if (!offset) {
  20.
  21.
               if ((carveout heap->total size -
  22.
                     carveout_heap->allocated_bytes) >= size
                                                                                                 关闭
  23.
                   pr_debug("%s: heap %s has enough memory (%1x) but"
  24.
                        " the allocation of size %lx still failed."
                       " Memory is probably fragmented.",
  26.
                        __func__, heap->name,
                       carveout_heap->total_size -
  27.
                       carveout_heap->allocated_bytes, size);
  28.
  29.
               return ION_CARVEOUT_ALLOCATE_FAIL;
  30.
  31.
            /*已经分配掉的内存字节。*/
  32.
           carveout_heap->allocated_bytes += size;
  33.
           return offset;
      }</span>
  34.
```

第28页 共69页 2017年09月07日 04:09

```
同样地,对于heap type为ION_HEAP_TYPE_SYSTEM的分配函数是ion_system_heap_allocate。
       [cpp]
  Θ1
        static int ion_system_contig_heap_allocate(struct ion_heap *heap,
  02
                               struct ion_buffer *buffer,
  03.
                               unsigned long len,
  04.
                               unsigned long align,
  05.
                              unsigned long flags)
  06.
            /*通过kzalloc分配。*/
  0.7
  08.
           buffer->priv_virt = kzalloc(len, GFP_KERNEL);
           if (!buffer->priv_virt)
  09.
 10.
               return -ENOMEM;
           atomic_add(len, &system_contig_heap_allocated);
  11.
 12.
           return 0:
 13. }
  01.
        <span xmlns="http://www.w3.org</pre>
        /1999/xhtml" style="">static int ion_system_contig_heap_allocate(struct ion_heap *heap,
                              struct ion_buffer *buffer,
                              unsigned long len,
  03.
  04.
                              unsigned long align.
  05
                               unsigned long flags)
  06.
  07.
           /*通过kzalloc分配。*/
  08.
           buffer->priv_virt = kzalloc(len, GFP_KERNEL);
           if (!buffer->priv_virt)
  09.
               return - ENOMEM:
  10.
  11.
           atomic_add(len, &system_contig_heap_allocated);
  12.
            return 0;
  13.
其他的几种Heap type可自行研究,接着调用ion_buffer_add将buffer添加到dev的buffers树上去
  01.
       static void ion buffer add(struct ion device *dev.
  02.
                       struct ion buffer *buffer)
  03.
  04.
            struct rb_node **p = &dev->buffers.rb_node;
  05.
            struct rb_node *parent = NULL;
           struct ion_buffer *entry;
  06.
  07.
  08.
           while (*p) {
              parent = *p;
  09.
  10.
               entry = rb_entry(parent, struct ion_buffer, node);
  11.
               if (buffer < entry) {</pre>
  13.
                   p = \&(*p)->rb left:
               } else if (buffer > entry) {
  14.
  15.
                   p = &(*p) -> rb_right;
  16.
               } else {
  17.
                    pr_err("%s: buffer already found.", __func__);
  18.
  19.
               }
  20.
       /*又是使用红黑树哦!*/
  21.
  22.
            rb_link_node(&buffer->node, parent, p);
  23.
            rb_insert_color(&buffer->node, &dev->buffers);
                                                                                                   关闭
       [cpp]
       <span xmlns="http://www.w3.org</pre>
  01.
        /1999/xhtml" style="">static void ion_buffer_add(struct ion_device *dev,
  02.
                       struct ion_buffer *buffer)
  03.
  04.
            struct rb_node **p = &dev->buffers.rb_node;
           struct rb_node *parent = NULL;
           struct ion_buffer *entry;
  06.
  07.
           while (*p) {
  08.
  09.
               parent = *p;
  10.
               entry = rb_entry(parent, struct ion_buffer, node);
  11.
               if (buffer < entry) {</pre>
```

第29页 共69页 2017年09月07日 04:09

```
13.
                                       p = &(*p)->rb_left;
    14.
                               } else if (buffer > entry) {
    15.
                                      p = &(*p) -> rb_right;
                               } else {
    16.
                                      pr_err("%s: buffer already found.", __func__);
   17.
    18.
    19.
    20.
               /*又是使用红黑树哦!*/
    21.
                       rb_link_node(&buffer->node, parent, p);
    23.
                       rb_insert_color(&buffer->node, &dev->buffers);
    24. }</span>
至此,已经得到client和handle,buffer分配完成!
ION IOC MAP/ ION IOC SHARE
               [cpp]
    01.
                int ion_share_dma_buf(struct ion_client *client, struct ion_handle *handle)
    02.
    03.
                        struct ion_buffer *buffer;
    04.
                        struct dma_buf *dmabuf;
    05.
                       bool valid_handle;
                       int fd:
    06.
    07.
    08.
                       mutex_lock(&client->lock);
    09.
                        valid_handle = ion_handle_validate(client, handle);
                        mutex_unlock(&client->lock);
    11.
                       if (!valid handle) {
                               WARN(1, "%s: invalid handle passed to share.\n", __func__);
    12.
   13.
                               return -EINVAL:
    14.
    15.
    16.
                        buffer = handle->buffer;
                       ion_buffer_get(buffer);
    17.
    18.
                        /*牛成一个新的file描述符*/
                       dmabuf = dma_buf_export(buffer, &dma_buf_ops, buffer->size, 0_RDWR);
   19.
    20.
                       if (IS_ERR(dmabuf)) {
    21.
                               ion_buffer_put(buffer);
                               return PTR_ERR(dmabuf);
    23.
                        /*将file转换用户空间识别的fd描述符。*/
    24.
                       fd = dma_buf_fd(dmabuf, O_CLOEXEC);
    25.
    26.
                       if (fd < 0)
    27.
                               dma_buf_put(dmabuf);
    28.
    29.
                       return fd;
    30.
               struct dma_buf *dma_buf_export(void *priv, const struct dma_buf_ops *ops,
    31.
    32.
                                              size_t size, int flags)
    33.
    34.
                        struct dma_buf *dmabuf;
    35.
                       struct file *file;
    36.
                ~~snip
                       /*分配struct dma_buf.*/
    37.
    38.
                        dmabuf = kzalloc(sizeof(struct dma_buf), GFP_KERNEL);
    39.
                       if (dmabuf == NULL)
    40.
                               return ERR_PTR(-ENOMEM);
                        /*保存信息到dmabuf,注意ops为dma_buf_ops,后面mmap为调用到。*/
    41.
    42.
                       dmabuf->priv = priv;
    43.
                        dmabuf->ops = ops;
    44.
                        dmabuf->size = size;
    45.
                        /*产生新的file*/
                        file = anon_inode_getfile("dmabuf", &dma_buf_fops, dmabuf, flags);
                                                                                                                                                                                                      关闭
    47.
    48.
                        dmabuf->file = file;
    49.
    50.
                        mutex_init(&dmabuf->lock);
    51.
                        INIT_LIST_HEAD(&dmabuf->attachments);
    53.
                        return dmabuf;
    54. }
    01.
                <span xmlns="http://www.w3.org</pre>
                \label{lem:client *client, struct ion_share_dma_buf(struct ion_client *client, struct ion_handle for the content of the cont
    02.
    03.
                       struct ion_buffer *buffer;
```

第30页 共69页 2017年09月07日 04:09

```
04.
           struct dma buf *dmabuf;
  05.
           bool valid_handle;
  06.
  07.
           mutex lock(&client->lock);
  08.
  09.
           valid_handle = ion_handle_validate(client, handle);
  10.
           mutex_unlock(&client->lock);
  11.
           if (!valid_handle) {
               WARN(1, "%s: invalid handle passed to share.\n", __func__);
  12.
 13.
               return -EINVAL;
 14.
 15.
  16.
           buffer = handle->buffer;
  17.
           ion_buffer_get(buffer);
  18.
           /*生成一个新的file描述符*/
  19.
           dmabuf = dma_buf_export(buffer, &dma_buf_ops, buffer->size, O_RDWR);
           if (IS_ERR(dmabuf)) {
  20.
  21.
               ion_buffer_put(buffer);
  22.
               return PTR ERR(dmabuf);
  23.
  24.
           /*将file转换用户空间识别的fd描述符。*/
  25.
           fd = dma_buf_fd(dmabuf, O_CLOEXEC);
           if (fd < 0)
  26.
  27.
               dma_buf_put(dmabuf);
  28.
  29.
           return fd;
  30.
       struct dma_buf *dma_buf_export(void *priv, const struct dma_buf_ops *ops,
  31.
  32.
                       size t size, int flags)
  33.
  34.
           struct dma_buf *dmabuf;
  35.
           struct file *file;
  36.
        ~snip
           /*分配struct dma_buf.*/
  37.
           dmabuf = kzalloc(sizeof(struct dma_buf), GFP_KERNEL);
  38.
  39.
           if (dmabuf == NULL)
  40.
               return ERR_PTR(-ENOMEM);
  41.
           /*保存信息到dmabuf,注意ops为dma_buf_ops,后面mmap为调用到。*/
           dmabuf->priv = priv;
           dmabuf->ops = ops;
  43.
           dmabuf->size = size:
  44.
  45.
           /*产生新的file*/
  46.
           file = anon_inode_getfile("dmabuf", &dma_buf_fops, dmabuf, flags);
  47.
  48.
           dmabuf->file = file;
  49.
  50.
           mutex init(&dmabuf->lock);
  51.
           INIT LIST HEAD(&dmabuf->attachments):
  52.
  53.
           return dmabuf;
  54. }</span>
通过上述过程,用户空间就得到了新的fd,重新生成一个新的fd的目的是考虑了两个用户空间进程想共享这块heap
内存的情况。然后再对fd作mmap,相应地kernel空间就调用到了file的dma_buf_fops中的
dma_buf_mmap_internal。
  01.
       static const struct file_operations dma_buf_fops = {
  02.
           .release
                     = dma_buf_release,
  03.
                      = dma_buf_mmap_internal,
  04.
       };
       static int dma_buf_mmap_internal(struct file *file, struct vm_area_struct *vma)
  05.
  06.
                                                                                               关闭
           struct dma buf *dmabuf:
  07.
  08.
  09.
           if (!is_dma_buf_file(file))
               return -EINVAL;
  10.
  11.
 12.
           dmabuf = file->private data;
           /*检查用户空间要映射的size是否比目前dmabuf也就是当前heap的size
  13.
  14.
       还要大,如果是就返回无效。*/
  15.
           /* check for overflowing the buffer's size */
  16.
           if (vma->vm_pgoff + ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT) >
 17.
               dmabuf->size >> PAGE_SHIFT)
  18.
               return -EINVAL:
  19.
           /*调用的是dma_buf_ops 的mmap函数*/
  20.
           return dmabuf->ops->mmap(dmabuf, vma);
```

第31页 共69页 2017年09月07日 04:09

```
21.
      }
22.
23.
     struct dma_buf_ops dma_buf_ops = {
         .map_dma_buf = ion_map_dma_buf,
24.
         .unmap_dma_buf = ion_unmap_dma_buf,
25.
26.
          .mmap = ion_mmap,
27.
          .release = ion_dma_buf_release,
28.
         .begin_cpu_access = ion_dma_buf_begin_cpu_access,
29.
         .end_cpu_access = ion_dma_buf_end_cpu_access,
30.
         .kmap_atomic = ion_dma_buf_kmap,
31.
         .kunmap_atomic = ion_dma_buf_kunmap,
32.
          .kmap = ion_dma_buf_kmap,
33.
          .kunmap = ion_dma_buf_kunmap,
34.
35.
     static int ion_mmap(struct dma_buf *dmabuf, struct vm_area_struct *vma)
36.
          struct ion buffer *buffer = dmabuf->priv;
37.
38.
         int ret;
39.
40.
          if (!buffer->heap->ops->map_user) {
41.
             pr_err("%s: this heap does not define a method for mapping "
                     "to userspace\n", __func__);
42.
43.
             return -EINVAL:
44.
45.
46.
         mutex_lock(&buffer->lock);
          /* now map it to userspace */
          /*调用的是相应heap的map_user,如carveout_heap_ops 调用的是
48.
     ion_carveout_heap_map_user ,此函数就是一般的mmap实现,不追下去了。*/
49.
50.
         ret = buffer->heap->ops->map_user(buffer->heap, buffer, vma);
51.
53.
             mutex_unlock(&buffer->lock);
             pr_err("%s: failure mapping buffer to userspace\n",
54.
55.
                    __func__);
56.
         } else {
57.
             buffer->umap_cnt++;
              mutex_unlock(&buffer->lock);
58.
60.
             vma->vm ops = &ion vm ops;
61.
              ^{\star} move the buffer into the <code>vm_private_data</code> so we can access it
62.
              * from vma_open/close
63.
65.
              vma->vm_private_data = buffer;
66.
67.
         return ret:
68. }
01.
      <span xmlns="http://www.w3.org</pre>
      /1999/xhtml" style="">static const struct file_operations dma_buf_fops = {
         .release = dma_buf_release,
03.
                     = dma_buf_mmap_internal,
          .mmap
04.
     };
     static int dma_buf_mmap_internal(struct file *file, struct vm_area_struct *vma)
05.
06.
07.
         struct dma buf *dmabuf;
09.
         if (!is_dma_buf_file(file))
10.
             return -EINVAL:
11.
12.
         dmabuf = file->private_data;
13.
          /*检查用户空间要映射的size是否比目前dmabuf也就是当前heap的size
      还要大,如果是就返回无效。*/
         /* check for overflowing the buffer's size */
15.
         if (vma->vm_pgoff + ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT) >
16.
17.
              dmabuf->size >> PAGE_SHIFT)
18.
              return -EINVAL;
19.
          /*调用的是dma_buf_ops 的mmap函数*/
20.
         return dmabuf->ops->mmap(dmabuf, vma);
21.
      }
22.
23.
     struct dma_buf_ops dma_buf_ops = {
24.
         .map_dma_buf = ion_map_dma_buf,
25.
         .unmap_dma_buf = ion_unmap_dma_buf,
26.
         .mmap = ion_mmap,
27.
         .release = ion_dma_buf_release,
28.
          .begin_cpu_access = ion_dma_buf_begin_cpu_access,
```

第32页 共69页 2017年09月07日 04:09

```
29.
           .end_cpu_access = ion_dma_buf_end_cpu_access,
  30.
           .kmap_atomic = ion_dma_buf_kmap,
           .kunmap_atomic = ion_dma_buf_kunmap,
  32.
           .kmap = ion dma buf kmap.
  33.
           .kunmap = ion_dma_buf_kunmap,
  34.
       };
  35.
       static int ion_mmap(struct dma_buf *dmabuf, struct vm_area_struct *vma)
  36.
  37.
           struct ion_buffer *buffer = dmabuf->priv;
  38.
           int ret:
  39.
  40.
           if (!buffer->heap->ops->map_user) {
               pr_err("%s: this heap does not define a method for mapping "
  41.
  42.
                      "to userspace\n", __func__);
  43.
               return -EINVAL;
  44.
  45.
  46.
           mutex_lock(&buffer->lock);
  47.
           /* now map it to userspace */
  48.
           /*调用的是相应heap的map_user,如carveout_heap_ops 调用的是
  49.
       ion_carveout_heap_map_user ,此函数就是一般的mmap实现,不追下去了。*/
  50.
           ret = buffer->heap->ops->map_user(buffer->heap, buffer, vma);
  51.
  52.
           if (ret) {
               mutex_unlock(&buffer->lock);
  53.
  54.
               pr_err("%s: failure mapping buffer to userspace\n",
  55.
                      __func__);
           } else {
  56.
               buffer->umap cnt++;
  57.
  58.
               mutex_unlock(&buffer->lock);
  59.
               vma->vm_ops = &ion_vm_ops;
  61.
                * move the buffer into the vm_private_data so we can access it
  62.
                * from vma_open/close
  63.
  64.
  65.
               vma->vm_private_data = buffer;
  66.
           return ret;
  68. }</span>
至此,用户空间就得到了bufferaddress,然后可以使用了!
ION IOC IMPORT
当用户空间另一个进程需要这块heap的时候,ION_IOC_IMPORT就派上用处了!注意,
传进去的fd为在ION IOC SHARE中得到的。
                   C Y
  01.
       struct ion handle *ion import dma buf(struct ion client *client, int fd)
  02.
  03.
  04.
           struct dma_buf *dmabuf;
           struct ion_buffer *buffer;
  06.
           struct ion_handle *handle;
  07.
           dmabuf = dma_buf_get(fd);
  08.
  09.
           if (IS_ERR_OR_NULL(dmabuf))
  10.
               return ERR_PTR(PTR_ERR(dmabuf));
  11.
           /* if this memory came from ion */
       ~~snip
  12.
          buffer = dmabuf->priv;
  13.
  14.
  15.
           mutex_lock(&client->lock);
  16.
           /^{\star} if a handle exists for this buffer just take \epsilon
                                                                                                关闭
       /*查找是否已经存在对应的handle了,没有则创建。因为另外一个进程只是
  18.
       调用了open 接口,对应的只创建了client,并没有handle。
  19.
           handle = ion handle lookup(client, buffer);
  20.
  21.
           if (!IS_ERR_OR_NULL(handle)) {
  22.
               ion_handle_get(handle);
  23.
  24.
  25.
           handle = ion handle create(client, buffer);
  26.
           if (IS ERR OR NULL(handle))
               goto end:
  27.
  28.
           ion_handle_add(client, handle);
  29.
```

第33页 共69页 2017年09月07日 04:09

mutex\_unlock(&client->lock);

```
31.
          dma_buf_put(dmabuf);
32.
          return handle;
33. }
     <span xmlns="http://www.w3.org</pre>
01.
      /1999/xhtml" style="">struct ion handle *ion import dma buf(struct ion client *client.:
0.2
03.
04.
          struct dma_buf *dmabuf;
         struct ion_buffer *buffer;
05.
         struct ion_handle *handle;
06.
07.
08.
         dmabuf = dma buf get(fd);
09.
         if (IS_ERR_OR_NULL(dmabuf))
10.
             return ERR_PTR(PTR_ERR(dmabuf));
         /* if this memory came from ion */
11.
12.
     ~~snip
         buffer = dmabuf->priv:
13.
14.
15.
         mutex_lock(&client->lock);
16.
          ^{\prime \star} if a handle exists for this buffer just take a reference to it ^{\star\prime}
      /*查找是否已经存在对应的handle了,没有则创建。因为另外一个进程只是
17.
      调用了open 接口,对应的只创建了client,并没有handle。
18.
19.
20.
         handle = ion_handle_lookup(client, buffer);
21.
          if (!IS_ERR_OR_NULL(handle)) {
22.
             ion_handle_get(handle);
23.
24.
         handle = ion handle create(client, buffer):
25.
26.
         if (IS_ERR_OR_NULL(handle))
27.
             goto end;
28.
         ion_handle_add(client, handle);
29.
30.
         mutex unlock(&client->lock);
31.
         dma buf put(dmabuf);
32.
         return handle;
33.
     }</span>
```

这样,用户空间另一个进程也得到了对应的bufferHandle,client/buffer/handle之间连接起来了!然后另一个一个进程就也可以使用mmap来操作这块heap buffer了。

和一般的进程使用ION区别就是共享的进程之间struction\_buffer是共享的,而struct ion\_handle是各自的。可见,ION的使用流程还是比较清晰的。不过要记得的是,使用好了ION,一定要释放掉,否则会导致内存泄露。

### ION内核空间使用

内核空间使用ION也是大同小异,按照创建client,buffer,handle的流程,只是它的使用对用户空间来说是透明的罢了!

ion\_client\_create在kernel空间被Qualcomm给封装了下。

```
| Composition | Paragraphic |
```

调用的流程也类似,不过map的时候调用的是heap对应的map\_kernel()而不是map\_user().

msm\_ion\_client\_create -> ion\_alloc ->ion\_map\_kernel

# 参考文档:

http://lwn.net/Articles/480055/

《ARM体系结构与编程》存储系统章节。

第34页 共69页 2017年09月07日 04:09

```
每个buffer一个handle,便于更灵活地细粒度地控制每个buffer的使用周期;
向用户进程输出fd,细粒度地对每个buffer进行mmap;
使用struct file可以重用已有struct file_operations进行mmap:
在binder driver中以BINDER_TYPE_FD类型为不同进程传递提供支撑,并借助fget/fput从struct file级别进行kref控制;
当不需要在用户态访问时,是不需要与struct file关联的,<mark>内核结构ion_handle/ion_buffer唯一的表征了该buffer</mark>,所以与struct file关
联的工作是在ioctl(ion, ION_IOC_SHARE/ION_ION_MAP, &share)中完成并输出的,用于后续的mmap调用;或者该进程不需要
mmap而是仅仅向别的进程binder transfer,这就实现了用户态进行buffer流转控制,而内核态完成buffer数据流转。
转自http://blog.csdn.net/kris_fei/article/details/8588661 & http://blog.csdn.net/kris_fei/article/details/8588661 & http://blog.csdn.net/kris_fei/article/details/8588661
考察平台:
chipset: MSM8X25Q
codebase: Android 4.1
ION概念:
ION是Google的下一代内存管理器,用来支持不同的内存分配机制,如CARVOUT(PMEM),物理连续内在
(kmalloc), 虚拟地址连续但物理不连续内存(vmalloc), IOMMU等。
用户空间和内核空间都可以使用ION,用户空间是通过/dev/ion来创建client的。
说到client, 顺便看下ION相关比较重要的几个概念。
Heap: 用来表示内存分配的相关信息,包括id, type, name等。用struct ion_heap表示。
Client: Ion的使用者,用户空间和内核控件要使用ION的buffer,必须先创建一个client,一个client可以有多个
buffer, 用struct ion buffer表示。
Handle: 将buffer该抽象出来,可以认为ION用handle来管理buffer,一般用户直接拿到的是handle,而不是buffer。
用struct ion_handle表示。
heap类型:
由于ION可以使用多种memory分配机制,例如物理连续和不连续的,所以ION使用enum ion_heap_type表示。
                  CP
  01.
        * enum ion_heap_types - list of all possible types of heaps
       * @ION_HEAP_TYPE_SYSTEM: memory allocated via vmalloc
  03.
       * @ION_HEAP_TYPE_SYSTEM_CONTIG: memory allocated via kmalloc
  04.
       * @ION_HEAP_TYPE_CARVEOUT: memory allocated from a prereserved
  05.
  06.
                       carveout heap, allocations are physically
  07.
                       contiguous
       * @ION_HEAP_TYPE_IOMMU: IOMMU memory
       * @ION_HEAP_TYPE_CP: memory allocated from a prereserved
  09.
                     carveout heap, allocations are physically
  10.
  11.
                     contiguous. Used for content protection.
       * @ION_HEAP_TYPE_DMA:
  12.
                                    memory allocated via DMA API
  13.
       * @ION_HEAP_END:
                             helper for iterating over heaps
  14.
  15.
       enum ion heap type {
           ION_HEAP_TYPE_SYSTEM,
  16.
  17.
           ION HEAP TYPE SYSTEM CONTIG,
  18.
           ION_HEAP_TYPE_CARVEOUT,
  19.
           ION_HEAP_TYPE_IOMMU,
  20.
           ION_HEAP_TYPE_CP,
  21.
           ION_HEAP_TYPE_DMA,
           ION_HEAP_TYPE_CUSTOM, /* must be last so device specific heaps always
  22.
  23.
                       are at the end of this enum */
  24.
           ION_NUM_HEAPS,
  25. };
                                                                                             关闭
        * enum ion_heap_types - list of all possible types of heaps
  02.
        * @ION_HEAP_TYPE_SYSTEM: memory allocated via vmalloc
  03.
       * @ION_HEAP_TYPE_SYSTEM_CONTIG: memory allocated via kmalloc
  04
  05.
       * @ION HEAP TYPE CARVEOUT: memory allocated from a prereserved
                       carveout heap, allocations are physically
  06.
                       contiguous
  08.
       * @ION_HEAP_TYPE_IOMMU: IOMMU memory
       * @ION_HEAP_TYPE_CP: memory allocated from a prereserved
  09.
  10.
                     carveout heap, allocations are physically
  11.
                      contiguous. Used for content protection.
```

其handle纳入进程file desc空间而不是/dev/ion设备内单独的handle空间,方便之处如下:

第35页 共69页 2017年09月07日 04:09

```
12.
         * @ION HEAP TYPE DMA:
                                        memory allocated via DMA API
  13.
         * @ION_HEAP_END:
                                helper for iterating over heaps
  14.
  15.
       enum ion heap type {
           ION HEAP TYPE SYSTEM,
  16.
  17.
           ION_HEAP_TYPE_SYSTEM_CONTIG,
  18.
           ION HEAP TYPE CARVEOUT,
           ION_HEAP_TYPE_IOMMU,
  19.
  20.
           ION_HEAP_TYPE_CP,
  21.
           ION_HEAP_TYPE_DMA,
           ION_HEAP_TYPE_CUSTOM, /* must be last so device specific heaps always
  22.
  23.
                        are at the end of this enum */
  24.
           ION NUM HEAPS,
  25. };
代码中的注释很明确地说明了哪种type对应的是分配哪种memory。不同type的heap需要不同的me
不过都是用struct ion_heap_ops来表示的。如以下例子:
  01.
        static struct ion heap ops carveout heap ops = {
  02.
            .allocate = ion_carveout_heap_allocate,
  03.
            .free = ion_carveout_heap_free,
            .phys = ion_carveout_heap_phys,
            .map_user = ion_carveout_heap_map_user,
           .map kernel = ion carveout heap map kernel.
  06.
  07.
            .unmap_user = ion_carveout_heap_unmap_user,
  08.
            .unmap_kernel = ion_carveout_heap_unmap_kernel,
  09.
            .map_dma = ion_carveout_heap_map_dma,
  10.
            .unmap_dma = ion_carveout_heap_unmap_dma,
           .cache_op = ion_carveout_cache_ops,
  11.
  12.
            .print debug = ion carveout print debug.
  13.
            .map iommu = ion carveout heap map iommu
  14.
            .unmap_iommu = ion_carveout_heap_unmap_iommu,
  15.
  16.
  17.
        static struct ion_heap_ops kmalloc_ops = {
           .allocate = ion_system_contig_heap_allocate,
  18.
  19.
            .free = ion_system_contig_heap_free,
  20.
            .phys = ion_system_contig_heap_phys,
  21.
            .map_dma = ion_system_contig_heap_map_dma,
  22.
            .unmap_dma = ion_system_heap_unmap_dma,
           .map_kernel = ion_system_heap_map_kernel,
  24.
            .unmap_kernel = ion_system_heap_unmap_kernel,
  25.
            .map_user = ion_system_contig_heap_map_user,
  26.
            .cache_op = ion_system_contig_heap_cache_ops,
  27.
            .print_debug = ion_system_contig_print_debug,
  28.
           .map_iommu = ion_system_contig_heap_map_iommu,
  29.
            .unmap_iommu = ion_system_heap_unmap_iommu,
  30. };
  01.
        static struct ion heap ons carveout heap ons = {
  02.
            .allocate = ion_carveout_heap_allocate,
  03.
            .free = ion_carveout_heap_free,
            .phys = ion_carveout_heap_phys,
  05.
            .map_user = ion_carveout_heap_map_user,
  06.
            .map kernel = ion carveout heap map kernel,
  07.
            .unmap user = ion carveout heap unmap user,
  08.
            .unmap_kernel = ion_carveout_heap_unmap_kernel,
  09.
            .map_dma = ion_carveout_heap_map_dma,
  10.
            .unmap_dma = ion_carveout_heap_unmap_dma,
                                                                                                   关闭
  11.
            .cache_op = ion_carveout_cache_ops,
  12.
            .print debug = ion carveout print debug,
  13.
            .map iommu = ion carveout heap map iommu
  14.
            .unmap_iommu = ion_carveout_heap_unmap_iommu,
  15.
  16.
  17.
        static struct ion_heap_ops kmalloc_ops = {
           .allocate = ion_system_contig_heap_allocate,
  18.
  19.
            .free = ion system contiq heap free,
  20.
            .phys = ion_system_contig_heap_phys,
```

第36页 共69页 2017年09月07日 04:09

.map\_dma = ion\_system\_contig\_heap\_map\_dma,

.unmap\_dma = ion\_system\_heap\_unmap\_dma,
.map\_kernel = ion\_system\_heap\_map\_kernel,
.unmap\_kernel = ion\_system\_heap\_unmap\_kernel,

21.

22.

## Heap ID:

同一种type的heap上当然可以分为若该干个chunk供用户使用,所以ION又使用ID来区分了。例如在type为ION\_HEAP\_TYPE\_CARVEOUT的heap上,audio和display部分都需要使用,ION就用ID来区分。Heap id用enumion heap ids表示。

```
CP
     [cpp]
01.
       ^{\ast} These are the only ids that should be used for Ion heap ids.
02.
       ^{\star} The ids listed are the order in which allocation will be attempted
03.
      * if specified. Don't swap the order of heap ids unless you know what
      * Id's are spaced by purpose to allow new Id's to be inserted in-between (f
      * possible fallbacks)
07.
08.
09.
10.
     enum ion_heap_ids {
11.
         INVALID_HEAP_ID = -1,
12.
          ION_CP_MM_HEAP_ID = 8,
         ION_CP_MFC_HEAP_ID = 12,
13.
         ION CP WB HEAP ID = 16, /* 8660 only */
14.
         ION CAMERA HEAP ID = 20, /* 8660 only */
15.
16.
         ION_SF_HEAP_ID = 24,
17.
          ION_IOMMU_HEAP_ID = 25,
18.
         ION_QSECOM_HEAP_ID = 26,
         ION_AUDIO_HEAP_BL_ID = 27,
19.
         ION AUDIO HEAP ID = 28.
20.
21.
22.
         ION_MM_FIRMWARE_HEAP_ID = 29,
23.
         ION_SYSTEM_HEAP_ID = 30,
24.
         ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_SECURE flag */
25.
26. };
     [qqɔ]
01.
02.
      * These are the only ids that should be used for Ion heap ids.
      * The ids listed are the order in which allocation will be attempted
      * if specified. Don't swap the order of heap ids unless you know what
      * you are doing!
05.
      * Id's are spaced by purpose to allow new Id's to be inserted in-between (for
06.
      * possible fallbacks)
07.
08.
09.
     enum ion_heap_ids {
         INVALID_HEAP_ID = -1,
11.
12.
         ION CP MM HEAP ID = 8.
13.
         ION CP MFC HEAP ID = 12,
         ION\_CP\_WB\_HEAP\_ID = 16, /* 8660 only */
14.
15.
          ION\_CAMERA\_HEAP\_ID = 20, /* 8660 only */
          ION_SF_HEAP_ID = 24,
16.
         ION_IOMMU_HEAP_ID = 25,
17.
         ION OSECOM HEAP ID = 26.
18.
         ION AUDIO HEAP BL ID = 27,
19.
                                                                                                   关闭
20.
         ION_AUDIO_HEAP_ID = 28,
21.
         ION_MM_FIRMWARE_HEAP_ID = 29,
22.
23.
         ION_SYSTEM_HEAP_ID = 30,
24.
         ION_HEAP_ID_RESERVED = 31 /** Bit reserved for ION_SECURE flag */
25.
26. };
```

## Heap 定义:

了解了heaptype和id,看看如何被用到了,本平台使用的文件为board-qrd7627a.c,有如下定义:

第37页 共69页 2017年09月07日 04:09

```
C Y
     [cpp]
01.
      * These heaps are listed in the order they will be allocated.
02.
      ^{\ast} Don't swap the order unless you know what you are doing!
03.
04.
05.
     struct ion_platform_heap msm7627a_heaps[] = {
06.
07.
                  .id = ION_SYSTEM_HEAP_ID,
                  .type = ION_HEAP_TYPE_SYSTEM,
09.
                  .name = ION_VMALLOC_HEAP_NAME,
10.
11.
     #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
12.
             /* PMEM ADSP = CAMERA */
13.
             {
                  .id = ION_CAMERA_HEAP_ID,
                 .type = CAMERA_HEAP_TYPE,
15.
16.
                  .name = ION CAMERA HEAP NAME,
17.
                  .memory_type = ION_EBI_TYPE,
18.
                  .extra_data = (void *)&co_mm_ion_pdata,
19.
                  .priv = (void *)&ion_cma_device.dev,
              /* AUDIO HEAP 1*/
21.
22.
             {
23.
                  .id = ION_AUDIO_HEAP_ID,
24.
                  .type = ION_HEAP_TYPE_CARVEOUT,
25.
                  .name = ION_AUDIO_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_ion_pdata,
28.
             }.
             /* PMEM_MDP = SF */
29.
30.
31.
                  .id = ION_SF_HEAP_ID,
                 .type = ION_HEAP_TYPE_CARVEOUT,
33.
                 .name = ION_SF_HEAP_NAME,
                 .memory_type = ION_EBI_TYPE,
34.
35.
                  .extra_data = (void *)&co_ion_pdata,
36.
37.
             /* AUDIO HEAP 2*/
38.
             {
39.
                  .id
                       = ION AUDIO HEAP BL ID,
                  .type = ION HEAP TYPE CARVEOUT.
40.
41.
                  .name = ION AUDIO BL HEAP NAME.
42.
                  .memory_type = ION_EBI_TYPE,
43.
                  .extra_data = (void *)&co_ion_pdata,
                  .base = BOOTLOADER_BASE_ADDR,
45.
46.
     #endif
47.
48.
     };
01.
      * These heaps are listed in the order they will be allocated.
03.
      * Don't swap the order unless you know what you are doing!
04.
05.
     struct ion_platform_heap msm7627a_heaps[] = {
06.
07.
                  .id = ION_SYSTEM_HEAP_ID,
08.
                  .type = ION_HEAP_TYPE_SYSTEM,
09.
                  .name = ION_VMALLOC_HEAP_NAME,
10.
             },
     #ifdef CONFIG MSM MULTIMEDIA USE ION
11.
12.
             /* PMEM_ADSP = CAMERA */
13.
                                                                                                  关闭
             {
                  .id = ION_CAMERA_HEAP_ID,
15.
                 .type = CAMERA_HEAP_TYPE,
                  .name = ION CAMERA HEAP NAME.
16.
17.
                  .memory_type = ION_EBI_TYPE,
18.
                  .extra_data = (void *)&co_mm_ion_pdata,
19.
                  .priv = (void *)&ion_cma_device.dev,
20.
             },
21.
             /* AUDIO HEAP 1*/
22.
                  .id = ION_AUDIO_HEAP_ID,
23.
24.
                  .type = ION_HEAP_TYPE_CARVEOUT,
25.
                  .name = ION_AUDIO_HEAP_NAME,
                 .memory_type = ION_EBI_TYPE,
26.
                 .extra_data = (void *)&co_ion_pdata,
```

第38页 共69页 2017年09月07日 04:09

```
28.
29.
              /* PMEM_MDP = SF */
30.
                  .id = ION_SF_HEAP_ID,
31.
                 .type = ION_HEAP_TYPE_CARVEOUT,
32.
33.
                 .name = ION_SF_HEAP_NAME,
34.
                 .memory_type = ION_EBI_TYPE,
35.
                 .extra_data = (void *)&co_ion_pdata,
36.
                AUDIO HEAP 2*/
38.
             {
39.
                  .id
                       = ION_AUDIO_HEAP_BL_ID,
40.
                 .type = ION_HEAP_TYPE_CARVEOUT,
41.
                 .name = ION_AUDIO_BL_HEAP_NAME,
                 .memory_type = ION_EBI_TYPE,
                 .extra_data = (void *)&co_ion_pdata,
43.
                 .base = BOOTLOADER BASE ADDR.
44.
45.
46.
47.
     #endif
48. };
```

## ION Handle:

47.

union {

当Ion client分配buffer时,相应的一个唯一的handle也会被指定,当然client可以多次申请ion buffer。申请好buffer之后,返回的是一个ion handle,不过要知道Ion buffer才和实际的内存相关,包括size,address等信息。Struct ion\_handle和struct ion\_buffer如下:

```
C P
     [cpp]
01.
02.
      * ion_handle - a client local reference to a buffer
      * @ref: reference count
      * @client:
                     back pointer to the client the buffer resides in
04.
       * @buffer: pointer to the buffer
05.
      * @node: node in the client's handle rbtree

* @kmap_cnt: count of times this client has mapped to kernel
06.
07.
08.
                        count of times this client has mapped for dma
      * Modifications to node, map_cnt or mapping should be protected by the
10.
      ^{\star} lock in the client. Other fields are never changed after initialization.
11.
12.
13.
      struct ion_handle {
14.
         struct kref ref;
15.
         struct ion_client *client;
         struct ion_buffer *buffer;
16.
17.
         struct rb node node:
18.
         unsigned int kmap_cnt;
19.
         unsigned int iommu_map_cnt;
20.
21.
22.
      ^{\star} struct ion_buffer - metadata for a particular buffer
23.
       * @ref:
                    refernce count
24.
25.
      * @node:
                     node in the ion_device buffers tree
                     back pointer to the ion_device
                    back pointer to the heap the buffer came from
      * @heap:
                  buffer specific flags
      * @flags:
28.
       * @size:
                     size of the buffer
29.
      * @priv_virt:
30.
                         private data to the buffer representable as
31.
                a void *
      * @priv_phys:
                       private data to the buffer representable as
33.
                an ion_phys_addr_t (and someday a phys_addr_+)
      * @lock:
                   protects the buffers cnt fields
34.
      * @kmap_cnt:
35.
                         number of times the buffer is mapped to the kernel
       * @vaddr:
36.
                     the kenrel mapping if kmap_cnt is not zero
37.
      * @dmap_cnt: number of times the buffer is mapped for dma
38.
                         the sg table for the buffer if dmap_cnt is not zero
      * @sg_table:
40.
     struct ion buffer {
41.
         struct kref ref:
42.
          struct rb_node node;
43.
          struct ion_device *dev;
          struct ion_heap *heap;
44.
45.
         unsigned long flags;
46.
         size t size;
```

第39页 共69页 2017年09月07日 04:09

```
48.
               void *priv_virt;
  49.
               ion_phys_addr_t priv_phys;
  50.
  51.
           struct mutex lock:
 52.
           int kmap cnt:
  53.
           void *vaddr:
  54.
           int dmap_cnt;
           struct sg_table *sg_table;
  56.
           int umap_cnt;
 57.
           unsigned int iommu_map_cnt;
 58.
           struct rb_root iommu_maps;
 59.
           int marked;
  60. };
  01.
        * ion_handle - a client local reference to a buffer
  02.
 03.
        * @ref: reference count
        * @client: back pointer to the client the buffer resides in 
* @buffer: pointer to the buffer 
* @node: node in the client's handle retroo
        * @client:
 04.
 05.
  06.
        * @node:
                      node in the client's handle rbtree
                       count of times this client has mapped to kernel
  07.
        * @kmap_cnt:
        * @dmap_cnt:
                           count of times this client has mapped for dma
 09.
        ^{\star} Modifications to node, map_cnt or mapping should be protected by the
 10.
        ^{\star} lock in the client. Other fields are never changed after initialization.
 11.
 12.
       struct ion_handle {
 13.
 14.
           struct kref ref;
           struct ion_client *client;
 15.
           struct ion buffer *buffer:
 16.
 17.
           struct rb_node node;
 18.
           unsigned int kmap_cnt;
 19.
           unsigned int iommu_map_cnt;
 20.
 21.
 22.
        * struct ion_buffer - metadata for a particular buffer
 23.
 24.
        * @ref:
                    refernce count
        * @node:
                       node in the ion_device buffers tree
  26.
                       back pointer to the ion_device
        * @heap:
                      back pointer to the heap the buffer came from
 27.
        * @flags: buffer specific flags
 28.
         * @size:
 29.
                      size of the buffer
                          private data to the buffer representable as
  30.
        * @priv_virt:
  31.
        * @priv_phys:
                          private data to the buffer representable as
 33.
                  an ion phys addr t (and someday a phys addr t)
        * @lock:
                    protects the buffers cnt fields
 34.
        * @kmap_cnt:
 35.
                          number of times the buffer is mapped to the kernel
 36.
        * @vaddr:
                       the kenrel mapping if kmap_cnt is not zero
 37.
        38.
        * @sg_table:
                           the sg table for the buffer if dmap_cnt is not zero
 39.
 40.
       struct ion_buffer {
 41.
           struct kref ref;
  42.
           struct rb_node node;
  43.
           struct ion_device *dev;
           struct ion_heap *heap;
 45.
           unsigned long flags;
 46.
           size_t size;
 47.
           union {
 48.
               void *priv_virt;
               ion_phys_addr_t priv_phys;
           };
                                                                                                   关闭
 51.
           struct mutex lock;
 52.
           int kmap cnt:
           void *vaddr:
 53.
 54.
           int dmap_cnt;
           struct sg_table *sg_table;
           int umap_cnt;
 57.
           unsigned int iommu_map_cnt;
 58.
           struct rb_root iommu_maps;
 59.
           int marked;
 60. };
ION Client:
```

第40页 共69页 2017年09月07日 04:09

```
用户空间和内核空间都可以成为client,不过创建的方法稍稍有点区别,先了解下基本的操作流程吧。
内核空间:
先创建client:
                 C Y
  01.
       struct ion_client *ion_client_create(struct ion_device *dev,
  02.
                          unsigned int heap_mask,
                          const char *name)
  03.
  01.
       struct ion_client *ion_client_create(struct ion_device *dev,
                         unsigned int heap mask.
 02.
 03
                         const char *name)
heap_mask: 可以分配的heap type, 如carveout,system heap, iommu等。
高通使用msm ion client create函数封装了下。
有了client之后就可以分配内存:
                 CP
       struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
  02.
                      size_t align, unsigned int flags)
       [cpp]
       struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
  01.
 02.
                      size_t align, unsigned int flags)
flags: 分配的heap id.
有了handle也就是buffer之后就准备使用了,不过还是物理地址,需要map:
  01.
       void *ion_map_kernel(struct ion_client *client, struct ion_handle *handle,
  02.
                 unsigned long flags)
  01.
       void *ion_map_kernel(struct ion_client *client, struct ion_handle *handle,
                  unsigned long flags)
用户空间:
    用户空间如果想使用ION,也必须先要创建client,不过它是打开/dev/ion,实际上它最终也会调用
    不过和内核空间创建client的一点区别是,用户空间不能选择heap type(使用预订的heap id隐含heap
type),但是内核空间却可以。
    另外,用户空间是通过IOCTL来分配内存的,cmd为ION_IOC_ALLOC.
                 C Y
       [cpp]
     ion_fd = open("/dev/ion", 0_ RDONLY | 0_SYNC);
  02. ioctl(ion_fd, ION_IOC_ALLOC, alloc);
      ion_fd = open("/dev/ion", O_ RDONLY | O_SYNC);
  02. ioctl(ion_fd, ION_IOC_ALLOC, alloc);
                                                                                         关闭
alloc为struct ion_allocation_data,len是申请buffer的长度,flags是heap id。
       [cpp]
  01.
  02.
       * struct ion_allocation_data - metadata passed from userspace for allocations
       * @len: size of the allocation
       * @align: required alignment of the allocation
  05.
       * @flags: flags passed to heap
       * @handle: pointer that will be populated with a cookie to use to refer
  06.
  07.
              to this allocation
  08.
```

第41页 共69页 2017年09月07日 04:09

```
09.
        ^{\star} Provided by userspace as an argument to the ioctl
  10.
  11.
       struct ion_allocation_data {
  12.
          size t len:
 13.
           size t align:
  14.
           unsigned int flags;
  15.
           struct ion_handle *handle;
        * struct ion_allocation_data - metadata passed from userspace for allocations
  02.
       * @len: size of the allocation
  03.
       * @align: required alignment of the allocation
  04.
  05.
       * @flags: flags passed to heap
       * @handle: pointer that will be populated with a cookie to use to refer
              to this allocation
  08.
       * Provided by userspace as an argument to the ioctl
  09.
 10.
  11.
       struct ion_allocation_data {
  12.
          size_t len;
  13.
           size_t align;
  14.
          unsigned int flags:
           struct ion_handle *handle;
  15.
  16. };
分配好了buffer之后,如果用户空间想使用buffer,先需要mmap. ION是通过先调用IOCTL中的
ION_IOC_SHARE/ION_IOC_MAP来得到可以mmap的fd,然后再执行mmap得到bufferaddress.
然后,你也可以将此fd传给另一个进程,如通过binder传递。在另一个进程中通过ION_IOC_IMPORT这个IOCTL
来得到这块共享buffer了。
来看一个例子:
                  CP
       进程A:
  01.
       int ionfd = open("/dev/ion", O_RDONLY | O_DSYNC);
  02.
 03.
       alloc_data.len = 0x1000;
  04.
       alloc_data.align = 0x1000;
       alloc_data.flags = ION_HEAP(ION_CP_MM_HEAP_ID);
       rc = ioctl(ionfd,ION_IOC_ALLOC, &alloc_data);
  07.
       fd_data.handle = alloc_data.handle;
       rc = ioctl(ionfd,ION IOC SHARE,&fd data);
  08.
  09.
       shared_fd = fd_data.fd;
  10.
       进程B:
       fd_data.fd = shared_fd;
 13. rc = ioctl(ionfd, ION_IOC_IMPORT, &fd_data);
       [cpp]
  01.
       int ionfd = open("/dev/ion", O_RDONLY | O_DSYNC);
  02.
  03.
       alloc_data.len = 0x1000;
       alloc_data.align = 0x1000;
       alloc_data.flags = ION_HEAP(ION_CP_MM_HEAP_ID);
       rc = ioctl(ionfd, ION_IOC_ALLOC, &alloc_data);
       fd data.handle = alloc data.handle;
  07.
  08.
       rc = ioctl(ionfd,ION IOC SHARE,&fd data);
  09.
       shared_fd = fd_data.fd;
  10.
  12.
       fd_data.fd = shared_fd;
 13. rc = ioctl(ionfd, ION IOC IMPORT, &fd data);
从上一篇ION基本概念中,我们了解了heaptype, heap id, client, handle以及如何使用,本篇再从原理上分析下
ION的运作流程。
MSM8x25Q平台使用的是board-qrd7627.c,ION相关定义如下:
                C y
  01. /**
```

第42页 共69页 2017年09月07日 04:09

```
02.
       ^{\star} These heaps are listed in the order they will be allocated.
03.
       * Don't swap the order unless you know what you are doing!
05.
      struct ion_platform_heap msm7627a_heaps[] = {
06.
              {
07.
                  .id = ION SYSTEM HEAP ID.
08.
                  .type = ION_HEAP_TYPE_SYSTEM,
09.
                  .name = ION_VMALLOC_HEAP_NAME,
10.
11.
      #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
              /* PMEM ADSP = CAMERA */
12.
13.
              {
14.
                  .id = ION_CAMERA_HEAP_ID,
                  .type = CAMERA_HEAP_TYPE,
15.
                  .name = ION_CAMERA_HEAP_NAME,
16.
17.
                  .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_mm_ion_pdata,
18.
19.
                  .priv = (void *)&ion_cma_device.dev,
20.
21.
              /* AUDIO HEAP 1*/
22.
              {
23.
                  .id = ION_AUDIO_HEAP_ID,
                  .type = ION HEAP TYPE CARVEOUT.
24.
25.
                  .name = ION_AUDIO_HEAP_NAME,
26.
                  .memory_type = ION_EBI_TYPE,
27.
                  .extra_data = (void *)&co_ion_pdata,
28.
                 PMEM_MDP = SF */
29.
              {
30.
31.
                  .id = ION_SF_HEAP_ID,
32.
                  .type = ION_HEAP_TYPE_CARVEOUT,
33.
                  .name = ION_SF_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_ion_pdata,
35.
36.
              },
              /* AUDTO HEAP 2*/
37.
38.
39.
                  .id
                       = ION_AUDIO_HEAP_BL_ID,
40.
                  .type = ION_HEAP_TYPE_CARVEOUT,
41.
                  .name = ION_AUDIO_BL_HEAP_NAME,
                  .memory_type = ION_EBI_TYPE,
42.
43.
                  .extra_data = (void *)&co_ion_pdata,
44.
                  .base = BOOTLOADER_BASE_ADDR,
45.
46.
47.
      #endif
48.
49.
50.
      static struct ion_co_heap_pdata co_ion_pdata = {
51.
          .adjacent_mem_id = INVALID_HEAP_ID,
52.
          .align = PAGE_SIZE,
53.
54.
55.
      static struct ion_co_heap_pdata co_mm_ion_pdata = {
56.
          .adjacent_mem_id = INVALID_HEAP_ID,
57.
          .align = PAGE_SIZE,
58.
59.
      static u64 msm dmamask = DMA BIT MASK(32);
60.
61.
62.
      static struct platform_device ion_cma_device = {
63.
         .name = "ion-cma-device",
          .id = -1,
65.
          .dev = {
             .dma mask = &msm dmamask.
66.
                                                                                                   关闭
              .coherent_dma_mask = DMA_BIT_MASK(32),
67.
68.
69. };
02.
       * These heaps are listed in the order they will be allocated.
       ^{\star} Don't swap the order unless you know what you are doing!
03.
04.
05.
      struct ion_platform_heap msm7627a_heaps[] = {
06.
             {
07.
                  .id = ION_SYSTEM_HEAP_ID,
08.
                  .type = ION_HEAP_TYPE_SYSTEM,
                  .name = ION_VMALLOC_HEAP_NAME,
09.
```

第43页 共69页 2017年09月07日 04:09

```
10.
               },
  11.
       #ifdef CONFIG_MSM_MULTIMEDIA_USE_ION
  12.
              /* PMEM_ADSP = CAMERA */
  13.
               {
                   .id = ION_CAMERA_HEAP_ID,
  14.
  15.
                   .type = CAMERA_HEAP_TYPE,
  16.
                   .name = ION_CAMERA_HEAP_NAME,
  17.
                  .memory_type = ION_EBI_TYPE,
                   .extra_data = (void *)&co_mm_ion_pdata,
  18.
                  .priv = (void *)&ion_cma_device.dev,
  19.
  20.
              },
               /* AUDIO HEAP 1*/
  21.
  22.
  23.
                   .id = ION_AUDIO_HEAP_ID,
                  .type = ION_HEAP_TYPE_CARVEOUT,
  25.
                  .name = ION_AUDIO_HEAP_NAME,
  26.
                  .memory_type = ION_EBI_TYPE,
  27.
                   .extra_data = (void *)&co_ion_pdata,
  28.
  29.
              /* PMEM_MDP = SF */
  30.
               {
                   .id = ION_SF_HEAP_ID,
  31.
  32.
                  .type = ION_HEAP_TYPE_CARVEOUT,
  33.
                   .name = ION_SF_HEAP_NAME,
  34.
                   .memory_type = ION_EBI_TYPE,
  35.
                   .extra_data = (void *)&co_ion_pdata,
  36.
               /* AUDIO HEAP 2*/
  37.
  38.
               {
  39.
                   .id
                        = ION_AUDIO_HEAP_BL_ID,
  40.
                   .type = ION_HEAP_TYPE_CARVEOUT,
  41.
                  .name = ION_AUDIO_BL_HEAP_NAME,
  42.
                   .memory_type = ION_EBI_TYPE,
                  .extra_data = (void *)&co_ion_pdata,
  43.
                   .base = BOOTLOADER_BASE ADDR,
  44.
  45.
               },
  46.
  47.
       #endif
  48.
  49.
  50.
       static struct ion_co_heap_pdata co_ion_pdata = {
  51.
           .adjacent_mem_id = INVALID_HEAP_ID,
  52.
           .align = PAGE_SIZE,
  53.
  54.
  55.
       static struct ion co heap pdata co mm ion pdata = {
           .adjacent_mem_id = INVALID_HEAP_ID,
  56.
  57.
           .align = PAGE_SIZE,
  58.
  59.
  60.
       static u64 msm_dmamask = DMA_BIT_MASK(32);
  61.
       static struct platform_device ion_cma device = {
  62.
  63.
           .name = "ion-cma-device",
  64.
           .id = -1,
  65.
           .dev = {
  66.
              .dma_mask = &msm_dmamask,
               .coherent_dma_mask = DMA_BIT_MASK(32),
  67.
  68.
  69. };
Qualcomm提示了不要轻易调换顺序,因为后面代码处理是将顺序定死了的,一旦你调换了,代码就无法正常运
行了。
                                                                                              关闭
另外,本系统中只使用了ION_HEAP_TYPE_CARVEOUT和 ION_HEAP_TYPE_SYSTEM这两种neap type.
对于ION_HEAP_TYPE_CARVEOUT的内存分配,后面将会发现,其实就是之前讲述过的使用mem pool来分配
Platform device如下,在msm_ion.c中用到。
       static struct ion_platform_data ion_pdata = {
  01.
  02.
           .nr = MSM_ION_HEAP_NUM,
  03.
           .has_outer_cache = 1,
  04.
           .heaps = msm7627a_heaps,
  05. };
```

第44页 共69页 2017年09月07日 04:09

```
06.
  07.
       static struct platform_device ion_dev = {
  08.
           .name = "ion-msm",
  09.
           .id = 1,
           .dev = { .platform_data = &ion_pdata },
 10.
  11. };
  01.
       static struct ion_platform_data ion_pdata = {
           .nr = MSM_ION_HEAP_NUM,
  03.
           .has_outer_cache = 1,
  04.
           .heaps = msm7627a_heaps,
 05.
  06.
  07.
       static struct platform_device ion_dev = {
  08.
           .name = "ion-msm",
           .id = 1,
  10.
           .dev = { .platform_data = &ion_pdata },
  11. };
ION初始化
转到msm ion.c, ion.c的某些函数也被重新封装了下.万事都从设备匹配开始:
  01.
       static struct platform_driver msm_ion_driver = {
  02.
           .probe = msm ion probe,
  03.
           .remove = msm_ion_remove,
  04.
           .driver = { .name = "ion-msm" }
  05.
  06.
       static int __init msm_ion_init(void)
  07.
           /*调用msm ion probe */
  08.
  09.
           return platform_driver_register(&msm_ion_driver);
  10.
  11.
  12.
       static int msm_ion_probe(struct platform_device *pdev)
 13.
  14.
           /*即board-grd7627a.c中的ion_pdata_*/
  15.
           struct ion_platform_data *pdata = pdev->dev.platform_data;
  16.
           int err;
  17.
           int i;
  18.
           /*heap数量*/
  19.
           num_heaps = pdata->nr;
  20.
  21.
           /*分配struct ion heap */
  22.
           heaps = kcalloc(pdata->nr, sizeof(struct ion_heap *), GFP_KERNEL);
  23.
  24.
           if (!heaps) {
  25.
               err = -ENOMEM:
  26.
               qoto out;
  27.
  28.
           /*创建节点,最终是/dev/ion,供用户空间操作。*/
  29.
           idev = ion_device_create(NULL);
  30.
           if (IS_ERR_OR_NULL(idev)) {
  31.
               err = PTR ERR(idev):
               goto freeheaps:
  32.
  33.
  34.
           /*最终是根据adjacent_mem_id 是否定义了来分配相邻内存,
  35.
       我们没用到,忽略此函数。*/
  36.
           msm_ion_heap_fixup(pdata->heaps, num_heaps);
  37.
           /^{\star} create the heaps as specified in the board file ^{\star}/
  38.
  39.
           for (i = 0; i < num_heaps; i++) {</pre>
                                                                                                  关闭
  40.
               struct ion_platform_heap *heap_data = &pdata->heaps[1];
  41.
               msm_ion_allocate(heap_data);
  43.
```

第45页 共69页 2017年09月07日 04:09

heaps[i] = ion\_heap\_create(heap\_data);
if (IS\_ERR\_OR\_NULL(heaps[i])) {

/\*创建ion heap。\*/

continue:

} else {

heaps[i] = 0;

if (heap\_data->size)

heap\_data->has\_outer\_cache = pdata->has\_outer\_cache;

44.

45.

46.

48.

49.

50.

51.

52.

```
54.
                                    heap_data->base,
 55.
                                    heap_data->size);
 56.
 57.
                      pr_info("ION heap %s created\n",
 58.
                                    heap data->name):
 59.
 60.
              /*创建的heap添加到idev中,以便后续使用。*/
 61.
              ion_device_add_heap(idev, heaps[i]);
 62.
          /*检查heap之间是否有重叠部分*/
 63.
          check_for_heap_overlap(pdata->heaps, num_heaps);
 64.
 65.
          platform_set_drvdata(pdev, idev);
 66.
          return 0;
 67.
 68.
      freeheaps:
 69.
          kfree(heaps);
 70.
      out:
 71.
          return err;
 72.
 73.
 74.
      通过ion_device_create创建/dev/ion节点:
 75.
      struct ion_device *ion_device_create(long (*custom_ioctl)
 76.
                           (struct ion_client *client,
 77.
                            unsigned int cmd,
 78.
                            unsigned long arg))
 79.
 80.
          struct ion_device *idev;
 81.
          int ret;
 82.
 83.
          idev = kzalloc(sizeof(struct ion_device), GFP_KERNEL);
 84.
          if (!idev)
 85.
              return ERR_PTR(-ENOMEM);
 86.
           /*是个misc设备*/
 87.
          idev->dev.minor = MISC DYNAMIC MINOR;
 88.
          /*节点名字为ion*/
          idev->dev.name = "ion";
 89.
 90.
          /*fops为ion_fops,所以对应ion的操作都会调用ion_fops的函数指针。*/
 91.
          idev->dev.fops = &ion_fops;
          idev->dev.parent = NULL;
 93.
          ret = misc register(&idev->dev);
          if (ret) {
 94.
 95
              pr_err("ion: failed to register misc device.\n");
 96.
              return ERR_PTR(ret);
 97.
          /*创建debugfs目录,路径为/sys/kernel/debug/ion/*/
 98.
          idev->debug_root = debugfs_create_dir("ion", NULL);
 99.
          if (IS_ERR_OR_NULL(idev->debug_root))
100.
101.
              pr_err("ion: failed to create debug files.\n");
102.
103.
          idev->custom_ioct1 = custom_ioct1;
104.
          idev->buffers = RB_ROOT;
105.
          mutex init(&idev->lock);
106.
          idev->heaps = RB ROOT:
107.
          idev->clients = RB_ROOT;
108.
          /*在ion目录下创建一个check_leaked_fds文件,用来检查ion的使用是否有内存泄漏。如果申请了ion之后
      不需要使用却没有释放,就会导致memory leak.*/
          debugfs_create_file("check_leaked_fds", 0664, idev->debug_root, idev,
109.
110.
                      &debug leak fops);
111.
          return idev:
112.
113.
114.
      msm_ion_allocate:
115.
      static void msm_ion_allocate(struct ion_platform_heap *heap)
116.
117.
                                                                                                 关闭
118.
          if (!heap->base && heap->extra_data) {
119.
              unsigned int align = 0;
              switch (heap->type) {
120.
121.
              /*获取align参数*
              case ION HEAP TYPE CARVEOUT:
122.
123.
                  align =
                  ((struct ion_co_heap_pdata *) heap->extra_data)->align;
124.
125.
                  break;
126.
              /*此type我们没使用到。*/
127.
              case ION_HEAP_TYPE_CP:
128.
129.
                  struct ion_cp_heap_pdata *data =
130.
                       (struct ion_cp_heap_pdata *)
131.
                      heap->extra_data;
                  if (data->reusable) {
```

第46页 共69页 2017年09月07日 04:09

```
133.
                      const struct fmem_data *fmem_info =
134.
                          fmem_get_info();
135.
                       heap->base = fmem_info->phys;
136.
                      data->virt_addr = fmem_info->virt;
                      pr_info("ION heap %s using FMEM\n", heap->name);
137.
138.
                  } else if (data->mem_is_fmem) {
139.
                      const struct fmem_data *fmem_info =
140.
                          fmem_get_info();
141.
                      heap->base = fmem_info->phys + fmem_info->size;
142.
143.
                  align = data->align:
144.
                  break:
145.
146.
              default:
                  break;
148.
              if (align && !heap->base) {
149.
150.
                  /*获取heap的base address。*/
151.
                  heap->base = msm_ion_get_base(heap->size,
                                    heap->memory_type,
152.
153.
                                    align);
154.
                  if (!heap->base)
155.
                      pr_err("%s: could not get memory for heap %s "
156.
                          "(id %x)\n", __func__, heap->name, heap->id);
157.
158.
159.
160.
       static unsigned long msm_ion_get_base(unsigned long size, int memory_type,
161.
162.
                          unsigned int align)
163.
164.
           switch (memory_type) {
           /*我们定义的是ebi type,看见没,此函数在mem pool中分析过了。
165.
      原理就是使用Mempool 来管理分配内存。*/
166.
167.
          case ION EBI TYPE:
168.
              return allocate_contiguous_ebi_nomap(size, align);
169.
170.
           case ION_SMI_TYPE:
171.
              return allocate_contiguous_memory_nomap(size, MEMTYPE_SMI,
172.
                                  align);
              break:
173.
174.
          default:
175.
              pr_err("%s: Unknown memory type %d\n", __func__, memory_type);
176.
177.
178.
179.
      ion heap create:
180.
       struct ion_heap *ion_heap_create(struct ion_platform_heap *heap_data)
181.
182.
           struct ion_heap *heap = NULL;
183.
          /*根据Heap type调用相应的创建函数。*/
184.
          switch (heap data->type) {
185.
          case ION HEAP TYPE SYSTEM CONTIG:
186.
              heap = ion_system_contig_heap_create(heap_data);
187.
              break:
188.
          case ION_HEAP_TYPE_SYSTEM:
189.
              heap = ion_system_heap_create(heap_data);
190.
              break;
          case ION HEAP TYPE CARVEOUT:
191.
192.
             heap = ion_carveout_heap_create(heap_data);
193.
              break:
194.
          case ION_HEAP_TYPE_IOMMU:
195.
             heap = ion_iommu_heap_create(heap_data);
196.
              break;
          case ION_HEAP_TYPE_CP:
197.
                                                                                                  关闭
198.
              heap = ion_cp_heap_create(heap_data);
199.
200.
      #ifdef CONFIG_CMA
201.
          case ION_HEAP_TYPE_DMA:
              heap = ion cma heap create(heap data);
202.
203.
              break:
204
      #endif
205.
          default:
206.
             pr_err("%s: Invalid heap type %d\n", __func__,
207.
                     heap_data->type);
208.
              return ERR PTR(-EINVAL);
209.
210.
211.
          if (IS_ERR_OR_NULL(heap)) {
              pr_err("%s: error creating heap %s type %d base %lu size %u\n",
```

第47页 共69页 2017年09月07日 04:09

```
213.
                      __func__, heap_data->name, heap_data->type,
214.
                      heap_data->base, heap_data->size);
215.
              return ERR_PTR(-EINVAL);
216.
          -
/*保存Heap的name,id和私有数据。*/
217.
218.
          heap->name = heap_data->name;
219.
          heap->id = heap_data->id;
220.
          heap->priv = heap_data->priv;
221.
          return heap;
222. }
01.
      static struct platform driver msm ion driver = {
02.
          .probe = msm ion probe,
03.
          .remove = msm_ion_remove,
 04.
          .driver = { .name = "ion-msm" }
 05.
      };
 06.
      static int __init msm_ion_init(void)
07.
          /*调用msm ion probe */
08.
 09.
          return platform_driver_register(&msm_ion_driver);
10.
11.
12.
      static int msm ion probe(struct platform device *pdev)
13.
14.
          /*即board-grd7627a.c中的ion pdata */
15.
          struct ion_platform_data *pdata = pdev->dev.platform_data;
16.
          int err;
17.
          int i;
18.
          /*hean数量*/
19.
20.
          num_heaps = pdata->nr;
21.
          /*分配struct ion_heap */
22.
          heaps = kcalloc(pdata->nr, sizeof(struct ion_heap *), GFP_KERNEL);
24.
          if (!heaps) {
25.
              err = -ENOMEM;
26.
              goto out;
27.
28.
          /*创建节点,最终是/dev/ion,供用户空间操作。*/
 29.
          idev = ion_device_create(NULL);
          if (IS_ERR_OR_NULL(idev)) {
30.
              err = PTR ERR(idev);
31.
              qoto freeheaps;
32.
 33.
 34.
          /*最终是根据adjacent_mem_id 是否定义了来分配相邻内存,
      我们没用到,忽略此函数。*/
36.
          msm ion heap fixup(pdata->heaps, num heaps);
37.
          /^{\star} create the heaps as specified in the board file ^{\star}/
38.
39.
          for (i = 0; i < num_heaps; i++) {</pre>
 40.
              struct ion_platform_heap *heap_data = &pdata->heaps[i];
 41.
              msm_ion_allocate(heap_data);
42.
43.
44.
              heap_data->has_outer_cache = pdata->has_outer_cache;
 45.
              /*创建ion heap。*/
 46.
              heaps[i] = ion_heap_create(heap_data);
              if (IS_ERR_OR_NULL(heaps[i])) {
48.
                  heaps[i] = 0;
49.
                  continue:
50.
              } else {
51.
                  if (heap_data->size)
                      pr_info("ION heap %s created at %lx "
                           "with size %x\n", heap_data->name
                                                                                                   关闭
54.
                                    heap_data->base,
55.
                                    heap_data->size);
56.
                  else
 57.
                      pr_info("ION heap %s created\n",
 58.
                                    heap_data->name);
 60.
              /*创建的heap添加到idev中,以便后续使用。*/
61.
              ion_device_add_heap(idev, heaps[i]);
62.
 63.
          /*检查heap之间是否有重叠部分*/
 64.
          check_for_heap_overlap(pdata->heaps, num_heaps);
 65.
          platform_set_drvdata(pdev, idev);
 66.
          return 0;
 67.
```

第48页 共69页 2017年09月07日 04:09

```
68.
      freeheaps:
 69.
          kfree(heaps);
 70.
 71.
          return err:
 72.
      }
 73.
 74.
       通过ion device create创建/dev/ion节点:
 75.
       struct ion_device *ion_device_create(long (*custom_ioctl)
 76.
                           (struct ion_client *client,
 77.
                            unsigned int cmd,
                            unsigned long arg))
 78.
 79.
 80.
           struct ion_device *idev;
 81.
 82.
           idev = kzalloc(sizeof(struct ion_device), GFP_KERNEL);
 83.
 84.
          if (!idev)
 85.
              return ERR_PTR(-ENOMEM);
 86.
           /*是个misc设备*/
 87.
           idev->dev.minor = MISC_DYNAMIC_MINOR;
 88.
           /*节点名字为ion*/
          idev->dev.name = "ion";
 89.
           /*fops为ion_fops,所以对应ion的操作都会调用ion_fops的函数指针。*/
 90.
 91.
           idev->dev.fops = &ion_fops;
 92.
           idev->dev.parent = NULL;
 93.
           ret = misc_register(&idev->dev);
           if (ret) {
              pr_err("ion: failed to register misc device.\n");
 95.
              return ERR_PTR(ret);
 96.
 97.
 98.
           /*创建debugfs目录,路径为/sys/kernel/debug/ion/*/
 99.
           idev->debug_root = debugfs_create_dir("ion", NULL);
100.
           if (IS_ERR_OR_NULL(idev->debug_root))
101.
              pr err("ion: failed to create debug files.\n");
102.
103.
          idev->custom_ioctl = custom_ioctl;
104.
           idev->buffers = RB_ROOT;
105.
           mutex_init(&idev->lock);
106.
          idev->heaps = RB_ROOT;
107.
          idev->clients = RB ROOT;
           /*在ion目录下创建一个check_leaked_fds文件,用来检查Ion的使用是否有内存泄漏。如果申请了ion之后
108.
       不需要使用却没有释放,就会导致memory leak.*/
109.
           debugfs_create_file("check_leaked_fds", 0664, idev->debug_root, idev,
110.
                      &debug_leak_fops);
111.
           return idev;
112.
113.
114.
       msm_ion_allocate:
115.
       static void msm_ion_allocate(struct ion_platform_heap *heap)
116.
117.
118.
          if (!heap->base && heap->extra data) {
              unsigned int align = 0:
119.
120.
              switch (heap->type) {
121.
              /*获取align参数*/
122.
              case ION_HEAP_TYPE_CARVEOUT:
123.
124.
                  ((struct ion co heap pdata *) heap->extra data)->align;
125.
                  break:
              /*此type我们没使用到。*/
126.
127.
              case ION_HEAP_TYPE_CP:
128.
129.
                  struct ion_cp_heap_pdata *data =
130.
                       (struct ion_cp_heap_pdata *)
131.
                      heap->extra data;
                                                                                                 关闭
132.
                  if (data->reusable) {
133.
                       const struct fmem_data *fmem_info =
                          fmem_get_info();
135.
                      heap->base = fmem_info->phys;
                      data->virt_addr = fmem_info->virt;
136.
                      pr_info("ION heap %s using FMEM\n", heap->name);
137.
138.
                  } else if (data->mem is fmem) {
139.
                       const struct fmem_data *fmem_info =
140.
                          fmem_get_info();
141.
                      heap->base = fmem_info->phys + fmem_info->size;
142.
                  }
143.
                  align = data->align;
144.
                  break:
145.
              default:
```

第49页 共69页 2017年09月07日 04:09

```
147.
                   break;
 148.
               if (align && !heap->base) {
 150.
                   /*获取heap的base address。*/
                   heap->base = msm_ion_get_base(heap->size,
 151.
 152.
                                    heap->memory_type,
 153.
                                    align);
 154.
                   if (!heap->base)
 155.
                      pr_err("%s: could not get memory for heap %s "
 156.
                         "(id %x)\n", __func__, heap->name, heap->id);
 157.
 158.
           }
 159.
       }
 160.
 161.
       static unsigned long msm_ion_get_base(unsigned long size, int memory_type,
 162.
                          unsigned int align)
 163.
 164.
           switch (memory_type) {
 165.
           /*我们定义的是ebi type,看见没,此函数在mem pool中分析过了。
       原理就是使用Mempool 来管理分配内存。*/
 166.
 167.
          case ION_EBI_TYPE:
168.
             return allocate_contiguous_ebi_nomap(size, align);
 169.
              break:
 170.
           case ION_SMI_TYPE:
 171.
              return allocate_contiguous_memory_nomap(size, MEMTYPE_SMI,
 172.
173.
174.
           default:
               pr_err("%s: Unknown memory type %d\n", __func__, memory_type);
175.
 176.
 177.
 178.
 179.
       ion_heap_create:
       struct ion_heap *ion_heap_create(struct ion_platform_heap *heap_data)
 180.
 181.
 182.
           struct ion_heap *heap = NULL;
 183.
           /*根据Heap type调用相应的创建函数。*/
           switch (heap_data->type) {
 184.
 185.
           case ION_HEAP_TYPE_SYSTEM_CONTIG:
 186.
              heap = ion_system_contig_heap_create(heap_data);
 187.
              break:
 188.
           case ION_HEAP_TYPE_SYSTEM:
 189.
              heap = ion_system_heap_create(heap_data);
 190.
 191.
          case ION_HEAP_TYPE_CARVEOUT:
             heap = ion_carveout_heap_create(heap_data);
 192.
 193.
               break:
 194.
           case ION HEAP TYPE IOMMU:
               heap = ion_iommu_heap_create(heap_data);
 195.
 196.
 197.
           case ION_HEAP_TYPE_CP:
 198.
               heap = ion_cp_heap_create(heap_data);
               break:
 199.
 200.
       #ifdef CONFIG CMA
 201.
           case ION_HEAP_TYPE_DMA:
 202.
               heap = ion_cma_heap_create(heap_data);
 203.
 204.
       #endif
 205.
           default:
 206
             pr_err("%s: Invalid heap type %d\n", __func__,
 207.
                     heap_data->type);
 208.
               return ERR_PTR(-EINVAL);
 209.
          }
 210.
          if (IS ERR OR NULL(heap)) {
 211.
                                                                                               关闭
              pr_err("%s: error creating heap %s type %d ba
 212.
 213.
                       _func__, heap_data->name, heap_data->type,
                     heap_data->base, heap_data->size);
 215.
               return ERR_PTR(-EINVAL);
 216.
           /*保存Heap的name,id和私有数据。*/
 217.
 218.
           heap->name = heap data->name:
 219.
           heap->id = heap_data->id;
 220.
           heap->priv = heap_data->priv;
 221.
           return heap;
 222. }
从下面的代码可以得知,ION_HEAP_TYPE_SYSTEM_CONTIG使用kmalloc创建
的,ION_HEAP_TYPE_SYSTEM使用的是vmalloc,而ion_carveout_heap_create就是系统预分配了一片内存区域
```

第50页 共69页 2017年09月07日 04:09

```
供其使用。Ion在申请使用的时候,会根据当前的type来操作各自的heap->ops。分别看下三个函数:
       [cpp]
  Θ1
       struct ion_heap *ion_system_contig_heap_create(struct ion_platform_heap *pheap)
  02.
  03.
           struct ion_heap *heap;
  04.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
  05.
  06.
           if (!heap)
  0.7
               return ERR PTR(-ENOMEM);
           /*使用的是kmalloc_ops,上篇有提到哦*/
  08.
           heap->ops = &kmalloc_ops;
  09.
  10.
           heap->type = ION_HEAP_TYPE_SYSTEM_CONTIG;
           system_heap_contig_has_outer_cache = pheap->has_outer_cache;
  11.
  12.
           return hean:
  13
  14.
       struct ion_heap *ion_system_heap_create(struct ion_platform_heap *pheap)
  15.
  16.
           struct ion_heap *heap;
  17.
           heap = kzalloc(sizeof(struct ion heap), GEP KERNEL):
  18.
  19.
           if (!heap)
  20.
               return ERR PTR(-ENOMEM);
  21.
           /*和上面函数的区别仅在于ops*,
           heap->ops = &vmalloc_ops;
  23.
           heap->type = ION HEAP TYPE SYSTEM:
  24.
           system heap has outer cache = pheap->has outer cache;
  25.
           return heap;
  26.
  27.
       struct ion_heap *ion_carveout_heap_create(struct ion_platform_heap *heap_data)
  28.
           struct ion carveout heap *carveout heap;
  29.
  30.
           int ret:
  31.
  32.
           carveout_heap = kzalloc(sizeof(struct ion_carveout_heap), GFP_KERNEL);
  33.
           if (!carveout_heap)
               return ERR_PTR(-ENOMEM);
           /* 重新创建一个新的pool,这里有点想不通的是为什么不直接使用全局的mempools呢?*/
  35.
           carveout_heap->pool = gen_pool_create(12, -1);
  36.
  37.
           if (!carveout heap->pool) {
  38.
               kfree(carveout_heap);
  39.
               return ERR_PTR(-ENOMEM);
  40.
           carveout_heap->base = heap_data->base;
  41.
  42.
           ret = gen pool add(carveout heap->pool, carveout heap->base,
  43.
                  heap_data->size, -1);
  44.
           if (ret < 0) {
  45.
               gen_pool_destroy(carveout_heap->pool);
               kfree(carveout_heap);
  47.
               return ERR_PTR(-EINVAL);
  48.
  49.
           carveout_heap->heap.ops = &carveout_heap_ops;
  50.
           carveout_heap->heap.type = ION_HEAP_TYPE_CARVEOUT;
           carveout_heap->allocated_bytes = 0;
           carveout_heap->total_size = heap_data->size;
           carveout_heap->has_outer_cache = heap_data->has_outer_cache;
  53.
  54.
  55.
           if (heap_data->extra_data) {
  56.
               struct ion_co_heap_pdata *extra_data =
  57.
                       heap_data->extra_data;
  59.
               if (extra data->setup region)
  60.
                   carveout_heap->bus_id = extra_data->setup_region();
  61.
               if (extra_data->request_region)
                                                                                                  关闭
  62.
                   carveout_heap->request_region =
                           extra_data->request_region;
               if (extra_data->release_region)
  65.
                   carveout_heap->release_region =
  66.
                           extra_data->release_region;
  67.
  68.
           return &carveout_heap->heap;
  69.
  71.
       Heap创建完成,然后保存到idev中:
  72.
       void ion_device_add_heap(struct ion_device *dev, struct ion_heap *heap)
  73.
  74.
           struct rb_node **p = &dev->heaps.rb_node;
           struct rb_node *parent = NULL;
  75.
```

第51页 共69页 2017年09月07日 04:09

```
76.
           struct ion_heap *entry;
 77.
 78.
           if (!heap->ops->allocate || !heap->ops->free || !heap->ops->map_dma ||
 79.
               !heap->ops->unmap dma)
              pr_err("%s: can not add heap with invalid ops struct.\n",
 80.
 81.
                      __func__);
 82.
          heap->dev = dev;
 83.
 84.
          mutex_lock(&dev->lock);
 85.
          while (*p) {
 86.
              parent = *p;
 87.
              entry = rb_entry(parent, struct ion_heap, node);
 88.
 89.
              if (heap->id < entry->id) {
                  p = &(*p)->rb_left;
 91.
              } else if (heap->id > entry->id ) {
                  p = &(*p)->rb_right;
 92.
 93.
              } else {
                  pr_err("%s: can not insert multiple heaps with "
 94.
 95.
                      "id %d\n", __func__, heap->id);
 96.
              }
 98.
           /*使用红里树保存*/
 99.
100.
           rb_link_node(&heap->node, parent, p);
101.
           rb_insert_color(&heap->node, &dev->heaps);
102.
           /*以heap name创建fs,位于ion目录下。如vamlloc, camera_preview , audio 等*/
103.
           debugfs_create_file(heap->name, 0664, dev->debug_root, heap,
104.
                      &debug_heap_fops);
105.
      end:
106.
           mutex_unlock(&dev->lock);
107.
 01.
       struct ion_heap *ion_system_contig_heap_create(struct ion_platform_heap *pheap)
 02.
 03.
           struct ion heap *heap;
 04.
 05.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
 06.
           if (!heap)
               return ERR_PTR(-ENOMEM);
           /*使用的是kmalloc_ops,上篇有提到哦*/
 08.
 09.
          heap->ops = &kmalloc ops;
 10.
          heap->type = ION_HEAP_TYPE_SYSTEM_CONTIG;
 11.
           {\tt system\_heap\_contig\_has\_outer\_cache = pheap->has\_outer\_cache;}
 12.
 13.
 14.
      struct ion heap *ion system heap create(struct ion platform heap *pheap)
 15.
 16.
           struct ion heap *heap;
 17.
 18.
           heap = kzalloc(sizeof(struct ion_heap), GFP_KERNEL);
 19.
          if (!heap)
              return ERR_PTR(-ENOMEM);
 20.
           /*和上面函数的区别仅在干ops*/
 21.
 22.
          heap->ops = &vmalloc_ops;
 23.
           heap->type = ION_HEAP_TYPE_SYSTEM;
 24.
           system_heap_has_outer_cache = pheap->has_outer_cache;
 25.
           return heap:
 26.
 27.
      struct ion_heap *ion_carveout_heap_create(struct ion_platform_heap *heap_data)
 28.
 29.
           struct ion_carveout_heap *carveout_heap;
 30.
                                                                                                   关闭
 31.
          carveout_heap = kzalloc(sizeof(struct ion_carveout_heap), GFP_KERNEL);
 32.
 33.
           if (!carveout_heap)
 34.
               return ERR PTR(-ENOMEM);
           /* 重新创建一个新的pool,这里有点想不通的是为什么不直接使用全局的mempools呢?*/
 35.
           carveout_heap->pool = gen_pool_create(12, -1);
 36.
           if (!carveout_heap->pool) {
 38.
              kfree(carveout_heap);
 39.
              return ERR_PTR(-ENOMEM);
 40.
 41.
           carveout_heap->base = heap_data->base;
 42.
          ret = gen_pool_add(carveout_heap->pool, carveout_heap->base,
 43.
                  heap_data->size, -1);
 44.
           if (ret < 0) {
              gen_pool_destroy(carveout_heap->pool);
 45.
```

第52页 共69页 2017年09月07日 04:09

```
46.
               kfree(carveout heap);
  47.
               return ERR_PTR(-EINVAL);
  48.
  49.
           carveout heap->heap.ops = &carveout heap ops:
           carveout heap->heap.type = ION HEAP TYPE CARVEOUT:
  50.
  51.
           carveout_heap->allocated_bytes = 0;
  52.
           carveout_heap->total_size = heap_data->size;
           carveout_heap->has_outer_cache = heap_data->has_outer_cache;
  55.
           if (heap_data->extra_data) {
               struct ion_co_heap_pdata *extra_data =
  56.
  57.
                      heap_data->extra_data;
  58.
               if (extra_data->setup_region)
  59.
                  carveout_heap->bus_id = extra_data->setup_region();
  61.
               if (extra data->request region)
  62.
                   carveout_heap->request_region =
  63.
                          extra_data->request_region;
  64.
               if (extra_data->release_region)
  65.
                   carveout_heap->release_region =
  66.
                          extra_data->release_region;
  67.
           return &carveout_heap->heap;
  68.
  69.
       }
  70.
  71.
       Heap创建完成,然后保存到idev中:
  72.
       void ion_device_add_heap(struct ion_device *dev, struct ion_heap *heap)
  73.
           struct rb_node **p = &dev->heaps.rb_node;
  74.
  75.
           struct rb_node *parent = NULL;
  76.
           struct ion_heap *entry;
  77.
  78.
           if (!heap->ops->allocate || !heap->ops->free || !heap->ops->map_dma ||
  79.
               !heap->ops->unmap dma)
               pr_err("%s: can not add heap with invalid ops struct.\n",
  80.
  81.
                      __func__);
  82.
  83.
           heap->dev = dev;
           mutex_lock(&dev->lock);
  85.
           while (*p) {
               parent = *p;
  86.
  87.
               entry = rb_entry(parent, struct ion_heap, node);
  88.
               if (heap->id < entry->id) {
  89.
  90.
                   p = &(*p)->rb_left;
               } else if (heap->id > entry->id ) {
  91.
  92.
                   p = &(*p)->rb_right;
  93.
               } else {
  94.
                   pr_err("%s: can not insert multiple heaps with "
  95.
                      "id %d\n", __func__, heap->id);
  97.
               }
  98.
           /*使用红里树保存*/
 99
 100.
           rb_link_node(&heap->node, parent, p);
 101.
           rb_insert_color(&heap->node, &dev->heaps);
           /*以heap name创建fs,位于ion目录下。如vamlloc, camera_preview , audio 等*/
 102.
           debugfs_create_file(heap->name, 0664, dev->debug_root, heap,
 103.
 104.
                      &debug_heap_fops);
 105.
       end:
 106.
           mutex_unlock(&dev->lock);
到此,ION初始化已经完成了。接下来该如何使用呢?嗯,通过前面创建的misc设备也就是idev了!还记得里面
有个fops为ion_fops吗?先来看下用户空间如何使用ION,最后看内核空
                                                                                                关闭
ION用户空间使用
       Ion_fops结构如下:
  01.
  02.
       static const struct file_operations ion_fops = {
```

第53页 共69页 2017年09月07日 04:09

```
10.
 11.
        static int ion_open(struct inode *inode, struct file *file)
 12.
            struct miscdevice *miscdev = file->private_data;
 13.
            struct ion_device *dev = container_of(miscdev, struct ion_device, dev);
 14.
            struct ion_client *client;
 15.
 16.
            char debug_name[64];
 17.
           pr_debug("%s: %d\n", __func__, __LINE__);
snprintf(debug_name, 64, "%u", task_pid_nr(current->group_leader));
 18.
 19.
            /*根据idev和task pid为name创建ion client*/
 20.
 21.
            client = ion_client_create(dev, -1, debug_name);
           if (IS_ERR_OR_NULL(client))
 22.
 23.
                return PTR_ERR(client);
            file->private_data = client;
 25.
            return 0:
 26.
 27. }
 01.
       Ion_fops结构如下:
        static const struct file_operations ion_fops = {
 02.
                          = THIS_MODULE,
 04.
                           = ion open,
            .open
 05.
                            = ion release.
            .release
 06.
            .unlocked_ioctl = ion_ioctl,
 07.
 08.
  09.
        用户空间都是通过ioctl来控制。先看ion_open.
 10.
        static int ion open(struct inode *inode, struct file *file)
 11.
 12.
 13.
            struct miscdevice *miscdev = file->private_data;
 14.
            struct ion_device *dev = container_of(miscdev, struct ion_device, dev);
            struct ion_client *client;
 15.
 16.
            char debug name[64]:
 17.
           pr_debug("%s: %d\n", __func__, __LINE__);
snprintf(debug_name, 64, "%u", task_pid_nr(current->group_leader));
 18.
 19.
 20.
            /*根据idev和task pid为name创建ion client*/
            client = ion_client_create(dev, -1, debug_name);
           if (IS_ERR_OR_NULL(client))
 22.
               return PTR ERR(client);
 23.
 24.
            file->private_data = client;
 25.
 26.
            return 0;
 27. }
前一篇文章有说到,要使用ION,必须要先创建ion client,因此用户空间在open ion的时候创建了client.
       struct ion_client *ion_client_create(struct ion_device *dev,
 01.
 02.
                             unsigned int heap_mask,
 03.
                             const char *name)
  04.
  05.
            struct ion_client *client;
            struct task_struct *task;
 06.
            struct rb node **p:
 07.
            struct rb_node *parent = NULL;
 08.
 09.
            struct ion_client *entry;
 10.
            pid_t pid;
 11.
            unsigned int name_len;
 12.
                                                                                                      关闭
            if (!name) {
 13.
                pr_err("%s: Name cannot be null\n", __func__);
 14.
 15.
                return ERR_PTR(-EINVAL);
 16.
 17.
            name_len = strnlen(name, 64);
 18.
 19.
            get task struct(current->group leader);
 20.
            task_lock(current->group_leader);
 21.
            pid = task_pid_nr(current->group_leader);
 22.
           /^{\star} don't bother to store task struct for kernel threads,
 23.
              they can't be killed anyway */
            if (current->group_leader->flags & PF_KTHREAD) {
 24.
                put task_struct(current->group_leader);
 25.
 26.
                task = NULL:
```

第54页 共69页 2017年09月07日 04:09

```
27.
          } else {
28.
              task = current->group_leader;
29.
30.
          task unlock(current->group leader):
          /*分配ion client struct.*/
31.
32.
          client = kzalloc(sizeof(struct ion_client), GFP_KERNEL);
33.
          if (!client) {
34.
             if (task)
35.
                  put_task_struct(current->group_leader);
36.
              return ERR_PTR(-ENOMEM);
37.
          }
          -
/*下面就是保存一系列参数了。*/
38.
39.
          client->dev = dev;
40.
          client->handles = RB_ROOT;
          mutex_init(&client->lock);
42.
          client->name = kzalloc(name_len+1, GFP_KERNEL);
43.
44.
          if (!client->name) {
45.
              put_task_struct(current->group_leader);
46.
              kfree(client);
47.
              return ERR_PTR(-ENOMEM);
48.
          } else {
49.
              strlcpy(client->name, name, name_len+1);
50.
51.
52.
          client->heap_mask = heap_mask;
53.
          client->task = task;
          client->pid = pid;
54.
55.
56.
          mutex_lock(&dev->lock);
57.
          p = &dev->clients.rb_node;
58.
          while (*p) {
59.
              parent = *p;
              entry = rb_entry(parent, struct ion_client, node);
60.
61.
62.
              if (client < entry)</pre>
63.
                 p = &(*p)->rb_left;
              else if (client > entry)
                  p = &(*p)->rb_right;
66.
          /*当前client添加到idev的clients根树上去。*/
67.
68.
          rb_link_node(&client->node, parent, p);
69.
          rb_insert_color(&client->node, &dev->clients);
70.
          /*在ION先创建的文件名字是以pid命名的。*/
71.
          client->debug_root = debugfs_create_file(name, 0664,
72.
                               dev->debug_root, client,
73.
74.
                               &debug_client_fops);
75.
          mutex_unlock(&dev->lock);
76.
          return client;
78. }
01.
      struct ion_client *ion_client_create(struct ion_device *dev,
02.
                           unsigned int heap_mask,
03.
                           const char *name)
04.
          struct ion_client *client;
06.
          struct task_struct *task;
          struct rb_node **p;
07.
          struct rb_node *parent = NULL;
08.
09.
          struct ion_client *entry;
          pid_t pid;
11.
                                                                                                   关闭
          unsigned int name_len;
12.
13.
          if (!name) {
14.
              pr_err("%s: Name cannot be null\n", __func__);
15.
              return ERR_PTR(-EINVAL);
16.
          name_len = strnlen(name, 64);
18.
19.
          get_task_struct(current->group_leader);
20.
          task_lock(current->group_leader);
21.
          pid = task_pid_nr(current->group_leader);
22.
          /st don't bother to store task struct for kernel threads,
23.
             they can't be killed anyway */
24.
          if (current->group_leader->flags & PF_KTHREAD) {
25.
              put_task_struct(current->group_leader);
```

第55页 共69页 2017年09月07日 04:09

```
26.
               task = NULL;
  27.
           } else {
  28.
               task = current->group_leader;
  29.
  30.
           task_unlock(current->group_leader);
  31.
           /*分配ion client struct.*/
  32.
           client = kzalloc(sizeof(struct ion_client), GFP_KERNEL);
  33.
           if (!client) {
  34.
               if (task)
  35.
                  put_task_struct(current->group_leader);
  36.
               return ERR_PTR(-ENOMEM);
  37.
           -
/*下面就是保存一系列参数了。*/
  38.
  39.
           client->dev = dev;
  40.
           client->handles = RB_ROOT;
  41.
           mutex init(&client->lock):
  42.
  43.
           client->name = kzalloc(name_len+1, GFP_KERNEL);
  44.
           if (!client->name) {
  45.
               put_task_struct(current->group_leader);
  46.
               kfree(client);
  47.
               return ERR_PTR(-ENOMEM);
  48.
           } else {
  49.
               strlcpy(client->name, name, name_len+1);
  50.
  51.
  52.
           client->heap_mask = heap_mask;
           client->task = task;
  53.
           client->pid = pid;
  54.
  55.
  56.
           mutex_lock(&dev->lock);
  57.
           p = &dev->clients.rb_node;
  58.
           while (*p) {
              parent = *p;
  59.
               entry = rb_entry(parent, struct ion_client, node);
  60.
  61.
  62.
               if (client < entry)</pre>
                   p = &(*p)->rb_left;
               else if (client > entry)
  65.
                  p = &(*p) -> rb_right;
  66.
  67.
           /*当前client添加到idev的clients根树上去。*/
  68.
           rb_link_node(&client->node, parent, p);
  69.
           rb_insert_color(&client->node, &dev->clients);
  70.
           /*在ION先创建的文件名字是以pid命名的。*/
  71.
  72.
           client->debug_root = debugfs_create_file(name, 0664,
  73.
                               dev->debug_root, client,
  74.
                               &debug_client_fops);
  75.
           mutex_unlock(&dev->lock);
  76.
  77.
           return client;
  78. }
有了client之后,用户程序就可以开始申请分配ION buffer了!通过ioctl命令实现。
ion ioct函数有若干个cmd, ION IOC ALLOC和ION IOC FREE相对应,表示申请和释放buffer。用户空间程序
使用前先要调用ION_IOC_MAP才能得到buffer address,而ION_IOC_IMPORT是为了将这块内存共享给用户空
间另一个进程。
  01.
       static long ion ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
  02.
  03.
           struct ion_client *client = filp->private_data;
                                                                                                关闭
  05.
           switch (cmd) {
           case ION_IOC_ALLOC:
  06.
  07.
  08.
               struct ion_allocation_data data;
  09.
  10.
               if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
                  return -EFAULT;
  11.
               /*分配buffer.*/
  12.
 13.
               data.handle = ion alloc(client, data.len, data.align,
  14.
                                data.flags);
  15.
               if (IS_ERR(data.handle))
                   return PTR_ERR(data.handle);
```

第56页 共69页 2017年09月07日 04:09

```
18.
19.
              if (copy_to_user((void __user *)arg, &data, sizeof(data))) {
20.
                 ion_free(client, data.handle);
21.
                  return -EFAULT;
22.
23.
             break:
24.
25.
         case ION_IOC_FREE:
26.
27.
              struct ion_handle_data data;
28.
             bool valid:
29.
30.
             if (copy_from_user(&data, (void __user *)arg,
31.
                         sizeof(struct ion_handle_data)))
                  return -EFAULT;
33.
             mutex_lock(&client->lock);
             valid = ion_handle_validate(client, data.handle);
34.
35.
              mutex_unlock(&client->lock);
36.
              if (!valid)
37.
                 return -EINVAL;
38.
              ion_free(client, data.handle);
39.
              break;
40.
41.
          case ION IOC MAP:
42.
          case ION_IOC_SHARE:
43.
              struct ion_fd_data data;
45.
             int ret;
             if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
46.
47.
                  return -EFAULT:
48.
              /*判断当前cmd是否被调用过了,调用过就返回,否则设置flags.*/
49.
              ret = ion_share_set_flags(client, data.handle, filp->f_flags);
50.
             if (ret)
51.
                  return ret;
52.
53.
              data.fd = ion_share_dma_buf(client, data.handle);
54.
              if (copy_to_user((void __user *)arg, &data, sizeof(data)))
55.
                  return -EFAULT;
              if (data.fd < 0)</pre>
57.
                 return data.fd;
              break:
58.
59.
60.
          case ION_IOC_IMPORT:
61.
62.
              struct ion_fd_data data;
              int ret = 0;
63.
             if (copy_from_user(&data, (void __user *)arg,
64.
65.
                        sizeof(struct ion_fd_data)))
66.
                 return -EFAULT;
67.
              data.handle = ion_import_dma_buf(client, data.fd);
             if (IS_ERR(data.handle))
69.
                  data.handle = NULL;
              if (copy_to_user((void __user *)arg, &data,
70.
71.
                      sizeof(struct ion_fd_data)))
72.
                  return -EFAULT;
73.
              if (ret < 0)
74.
                 return ret;
75.
              break;
76.
77.
         case ION_IOC_CUSTOM:
78.
     ~~snip
79.
         case ION_IOC_CLEAN_CACHES:
         case ION_IOC_INV_CACHES:
81.
         case ION_IOC_CLEAN_INV_CACHES:
82.
          ~~snip
                                                                                                   关闭
         case ION_IOC_GET_FLAGS:
83.
84.
      ~~snip
85.
         default:
86.
             return -ENOTTY;
87.
88.
          return 0;
89.
     }
01.
      static long ion_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
02.
03.
          struct ion_client *client = filp->private_data;
04.
05.
          switch (cmd) {
```

第57页 共69页 2017年09月07日 04:09

```
06.
          case ION IOC ALLOC:
07.
08.
              struct ion_allocation_data data;
09.
             if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
10.
11.
                  return -EFAULT;
              /*分配buffer.*/
12.
13.
              data.handle = ion_alloc(client, data.len, data.align,
14.
                               data.flags);
15.
             if (IS ERR(data.handle))
16.
17.
                  return PTR_ERR(data.handle);
18.
19.
              if (copy_to_user((void __user *)arg, &data, sizeof(data))) {
                  ion_free(client, data.handle);
21.
                  return -EFAULT;
22.
23.
              break:
24.
25.
         case ION_IOC_FREE:
26.
27.
              struct ion_handle_data data;
             bool valid:
28.
29.
30.
             if (copy_from_user(&data, (void __user *)arg,
31.
                        sizeof(struct ion_handle_data)))
                  return -EFAULT;
             mutex lock(&client->lock);
33.
             valid = ion_handle_validate(client, data.handle);
34.
35.
              mutex_unlock(&client->lock);
36.
              if (!valid)
37.
                  return -EINVAL;
38.
              ion_free(client, data.handle);
39.
              break;
40.
41.
          case ION IOC MAP:
42.
          case ION_IOC_SHARE:
43.
              struct ion_fd_data data;
45.
             int ret;
             if (copy_from_user(&data, (void __user *)arg, sizeof(data)))
46.
47.
                  return -EFAULT:
48.
              /*判断当前cmd是否被调用过了,调用过就返回,否则设置flags.*/
49.
              ret = ion_share_set_flags(client, data.handle, filp->f_flags);
50.
             if (ret)
                  return ret;
51.
52.
53.
              data.fd = ion_share_dma_buf(client, data.handle);
54.
              if (copy_to_user((void __user *)arg, &data, sizeof(data)))
55.
                  return -EFAULT;
              if (data.fd < 0)</pre>
57.
                  return data.fd;
58.
              break:
59.
60.
          case ION_IOC_IMPORT:
61.
62.
              struct ion_fd_data data;
63.
              int ret = 0;
64.
             if (copy_from_user(&data, (void __user *)arg,
65.
                        sizeof(struct ion_fd_data)))
66.
                  return -EFAULT;
              data.handle = ion_import_dma_buf(client, data.fd);
             if (IS_ERR(data.handle))
69.
                  data.handle = NULL;
              if (copy_to_user((void __user *)arg, &data,
70.
                                                                                                   关闭
71.
                       sizeof(struct ion_fd_data)))
72.
                  return -EFAULT;
73.
              if (ret < 0)
                  return ret;
              break;
75.
76.
77.
         case ION IOC CUSTOM:
78.
     ~~snip
79.
         case ION_IOC_CLEAN_CACHES:
80.
         case ION_IOC_INV_CACHES:
         case ION_IOC_CLEAN_INV_CACHES:
81.
82.
          ~~snip
83.
         case ION IOC GET FLAGS:
84.
     ~~snip
          default:
```

第58页 共69页 2017年09月07日 04:09

```
86.
               return -ENOTTY;
  87.
  88.
            return 0;
  89.
下面分小节说明分配和共享的原理。
ION IOC ALLOC
                  C P
       [qqɔ]
  01.
        struct ion_handle *ion_alloc(struct ion_client *client, size_t len,
  02.
                        size_t align, unsigned int flags)
  04.
        ~~snip
  05.
  06.
           mutex lock(&dev->lock):
  07.
            /*循环遍历当前Heap链表。*/
  08.
           for (n = rb_first(&dev->heaps); n != NULL; n = rb_next(n)) {
               struct ion_heap *heap = rb_entry(n, struct ion_heap, node);
  09.
  10.
       /*只有heap type和id都符合才去创建buffer.*/
               /* if the client doesn't support this heap type */
  11.
 12.
               if (!((1 << heap->type) & client->heap_mask))
  13.
                   continue;
               /^{\star} if the caller didn't specify this heap type ^{\star}/
  14.
               if (!((1 << heap->id) & flags))
  16.
                   continue;
               /* Do not allow un-secure heap if secure is specified */
  17.
 18.
               if (secure_allocation && (heap->type != ION_HEAP_TYPE_CP))
  19.
                   continue:
  20.
               buffer = ion_buffer_create(heap, dev, len, align, flags);
  21.
         ~snip
  22.
  23.
           mutex unlock(&dev->lock):
  24.
  25.
  26.
            /*创建了buffer之后,就相应地创建handle来管理buffer.*/
           handle = ion_handle_create(client, buffer);
  28.
  29.
        ~~snin
  30.
       }
  31.
  32.
        找到Heap之后调用ion_buffer_create:
  33.
        static struct ion_buffer *ion_buffer_create(struct ion_heap *heap,
  34.
                            struct ion_device *dev,
  35.
                            unsigned long len,
  36.
                            unsigned long align
  37.
                            unsigned long flags)
  38.
            struct ion_buffer *buffer;
  40.
            struct sq table *table;
  41.
           int ret;
            /*分配struct ion buffer,用来管理buffer.*/
  42.
  43.
           buffer = kzalloc(sizeof(struct ion_buffer), GFP_KERNEL);
  44.
            if (!buffer)
  45.
               return ERR_PTR(-ENOMEM);
  46.
           buffer->heap = heap;
  47.
  48.
            kref_init(&buffer->ref);
  49.
            /*调用相应heap type的ops allocate。还记得前面有提到过不同种类的ops吗,
  50.
        如carveout_heap_ops , vmalloc_ops 。*/
           ret = heap->ops->allocate(heap, buffer, len, align, flags);
  52.
           if (ret) {
  53.
               kfree(buffer);
  54.
               return ERR_PTR(ret);
                                                                                                  关闭
  55.
  56.
            buffer->dev = dev;
            buffer->size = len;
  58.
  59.
            /*http://lwn.net/Articles/263343/*/
  60.
            table = buffer->heap->ops->map_dma(buffer->heap, buffer);
  61.
            if (IS_ERR_OR_NULL(table)) {
  62.
               heap->ops->free(buffer);
               kfree(buffer);
               return ERR_PTR(PTR_ERR(table));
  64.
  65.
  66.
           buffer->sg_table = table;
  67.
  68.
            mutex_init(&buffer->lock);
```

第59页 共69页 2017年09月07日 04:09

```
69.
         /*将当前ion buffer添加到idev 的buffers 树上统一管理。*/
70.
         ion_buffer_add(dev, buffer);
71.
         return buffer;
72.
     [cpp]
01.
     struct ion handle *ion alloc(struct ion client *client, size t len.
0.2
                      size t align, unsigned int flags)
03.
04.
      ~~snip
05.
06.
         mutex_lock(&dev->lock);
         /*循环谝历当前Hean辩表。*/
07.
08.
         for (n = rb_first(&dev->heaps); n != NULL; n = rb_next(n)) {
09.
             struct ion_heap *heap = rb_entry(n, struct ion_heap, node);
10.
     /*只有heap type和id都符合才去创建buffer.*/
11.
              /* if the client doesn't support this heap type */
12.
             if (!((1 << heap->type) & client->heap_mask))
13.
                 continue:
             /^{\star} if the caller didn't specify this heap type ^{\star}/
14.
15.
             if (!((1 << heap->id) & flags))
16.
              /* Do not allow un-secure heap if secure is specified */
17.
18.
             if (secure allocation && (heap->type != ION HEAP TYPE CP))
19.
                 continue:
20.
             buffer = ion_buffer_create(heap, dev, len, align, flags);
21.
     ~~snip
22.
23.
         mutex_unlock(&dev->lock);
24.
25.
         /*创建了buffer之后,就相应地创建handle来管理buffer.*/
26.
27.
         handle = ion_handle_create(client, buffer);
28.
29.
30.
     }
31.
     找到Heap之后调用ion_buffer_create:
32.
33.
      static struct ion_buffer *ion_buffer_create(struct ion_heap *heap,
34.
                          struct ion_device *dev,
35.
                          unsigned long len,
                          unsigned long align,
36.
37.
                          unsigned long flags)
38.
39.
         struct ion_buffer *buffer;
40.
         struct sg_table *table;
42.
         /*分配struct ion buffer,用来管理buffer.*/
         buffer = kzalloc(sizeof(struct ion_buffer), GFP_KERNEL);
43.
44.
         if (!buffer)
45.
              return ERR_PTR(-ENOMEM);
46.
47.
         buffer->heap = heap;
         kref_init(&buffer->ref);
48.
         /*调用相应heap type的ops allocate。还记得前面有提到过不同种类的ops吗,
49.
50.
     如carveout_heap_ops , vmalloc_ops 。*/
51.
         ret = heap->ops->allocate(heap, buffer, len, align, flags);
52.
         if (ret) {
             kfree(buffer);
54.
             return ERR_PTR(ret);
55.
56.
57.
         buffer->dev = dev;
          /*http://lwn.net/Articles/263343/*/
                                                                                                 关闭
         table = buffer->heap->ops->map_dma(buffer->heap, buffer);
60.
         if (IS_ERR_OR_NULL(table)) {
61.
62.
             heap->ops->free(buffer);
63.
             kfree(buffer);
             return ERR_PTR(PTR_ERR(table));
64.
66.
         buffer->sg_table = table;
67.
68.
         mutex init(&buffer->lock);
         /*将当前ion buffer添加到idev 的buffers 树上统一管理。*/
69.
70.
         ion_buffer_add(dev, buffer);
71.
72. }
```

第60页 共69页 2017年09月07日 04:09

```
C Y
       [cpp]
  01.
        static struct ion_handle *ion_handle_create(struct ion_client *client,
                struct ion_buffer *buffer)
  03.
  04.
        struct ion handle *handle:
  05.
        /*分配struct ion handle.*/
  06.
        handle = kzalloc(sizeof(struct ion_handle), GFP_KERNEL);
  07.
        if (!handle)
         return ERR_PTR(-ENOMEM);
  09.
        kref_init(&handle->ref);
 10.
        rb init node(&handle->node):
  11.
        handle->client = client; //client放入handle中
  12.
        ion_buffer_get(buffer); //引用计数加1
  13.
        handle->buffer = buffer; //buffer也放入handle中 return handle;
  14.
  15.
       [cpp]
       static struct ion_handle *ion_handle_create(struct ion_client *client,
  01.
  02.
                struct ion_buffer *buffer)
  03.
  04.
        struct ion_handle *handle;
  05.
        /*分配struct ion_handle.*
        handle = kzalloc(sizeof(struct ion_handle), GFP_KERNEL);
  07.
        if (!handle)
         return ERR PTR(-ENOMEM);
  08.
  09.
        kref_init(&handle->ref);
  10.
        rb_init_node(&handle->node);
        handle->client = client; //client放入handle中
  12.
        ion_buffer_get(buffer); //引用计数加1
        handle->buffer = buffer; //buffer也放入handle中 return handle;
  13.
  14.
  15.
       </n>
先拿heap type为ION_HEAP_TYPE_CARVEOUT为例,看下它是如何分配buffer的。
allocate对应ion_carveout_heap_allocate。
       [cpp]
  01.
       static int ion_carveout_heap_allocate(struct ion_heap *heap,
  02.
                             struct ion_buffer *buffer,
  03.
                             unsigned long size, unsigned long align,
  04.
                             unsigned long flags)
  05.
           buffer->priv phys = ion_carveout_allocate(heap, size, align);
  06.
  07.
           return buffer->priv_phys == ION_CARVEOUT_ALLOCATE_FAIL ? -ENOMEM : 0;
  08.
  09.
       ion\_phys\_addr\_t\ ion\_carveout\_allocate(\textbf{struct}\ ion\_heap\ *heap,
  10.
                             unsigned long size,
  11.
                             unsigned long align)
  12.
  13.
           struct ion_carveout_heap *carveout_heap =
  14.
               container_of(heap, struct ion_carveout_heap, heap);
           /*通过创建的mem pool来管理buffer,由于这块buffer在初始化的
  15
  16.
       时候就预留了,现在只要从上面拿一块区域就可以了。*/
  17.
           unsigned long offset = gen_pool_alloc_aligned(carveout_heap->pool,
  18.
                                   size, ilog2(align));
  19.
           /*分配不成功可能是没有内存空间可供分配了或者是有碎片导致的。*/
  20.
           if (!offset) {
  21.
               if ((carveout_heap->total_size -
                     carveout_heap->allocated_bytes) >= size)
                   pr_debug("%s: heap %s has enough memory (%lx) but"
  23.
                        the allocation of size %lx still fa
  24.
                                                                                                 关闭
                       " Memory is probably fragmented.",
  25.
  26.
                         _func__, heap->name,
  27.
                       carveout_heap->total_size -
  28.
                       carveout_heap->allocated_bytes, size);
               return ION_CARVEOUT_ALLOCATE_FAIL;
  29.
  30.
           /*已经分配掉的内存字节。*/
  31.
  32.
           carveout_heap->allocated_bytes += size;
  33.
  01. static int ion carveout heap allocate(struct ion heap *heap,
```

第61页 共69页 2017年09月07日 04:09

```
02.
                             struct ion buffer *buffer,
  03.
                             unsigned long size, unsigned long align,
  04.
                             unsigned long flags)
  05.
           buffer->priv_phys = ion_carveout_allocate(heap, size, align);
  06.
  07.
           return buffer->priv_phys == ION_CARVEOUT_ALLOCATE_FAIL ? -ENOMEM : 0;
  08.
  09.
       ion_phys_addr_t ion_carveout_allocate(struct ion_heap *heap,
  10.
                             unsigned long size,
  11.
                             unsigned long align)
  12.
           struct ion_carveout_heap *carveout_heap =
  13.
  14.
               container_of(heap, struct ion_carveout_heap, heap);
  15.
           /*通过创建的mem pool来管理buffer,由于这块buffer在初始化的
       时候就预留了,现在只要从上面拿一块区域就可以了。*/
  16.
  17.
           unsigned long offset = gen_pool_alloc_aligned(carveout_heap->pool,
 18.
                                  size, ilog2(align)):
           /*分配不成功可能是没有内存空间可供分配了或者是有碎片导致的。*/
  19.
  20.
           if (!offset) {
  21.
               if ((carveout_heap->total_size -
  22.
                     carveout_heap->allocated_bytes) >= size)
  23.
                   pr_debug("%s: heap %s has enough memory (%lx) but"
                        ' the allocation of size %lx still failed."
  24.
                       " Memory is probably fragmented.",
  25.
  26.
                        __func__, heap->name,
  27.
                       carveout_heap->total_size -
                       carveout_heap->allocated_bytes, size);
               return ION_CARVEOUT_ALLOCATE_FAIL;
  29.
  30.
           /*已经分配掉的内存字节。*/
  31.
  32.
           carveout_heap->allocated_bytes += size;
  33.
           return offset;
同样地,对于heap type为ION_HEAP_TYPE_SYSTEM的分配函数是ion_system_heap_allocate。
       [cpp]
  01.
       static int ion system contiq heap allocate(struct ion heap *heap,
  0.2
                              struct ion buffer *buffer,
  03.
                              unsigned long len,
  04.
                              unsigned long align,
  05.
                              unsigned long flags)
  06.
           /*诵讨kzalloc分配。*/
  07.
  08.
           buffer->priv_virt = kzalloc(len, GFP_KERNEL);
  09.
           if (!buffer->priv_virt)
  10.
               return -ENOMEM;
  11.
           atomic_add(len, &system_contig_heap_allocated);
  12.
           return 0;
  13.
  01.
       static int ion_system_contig_heap_allocate(struct ion_heap *heap,
  02.
                              struct ion_buffer *buffer,
  03.
                              unsigned long len,
  04.
                              unsigned long align,
  05.
                              unsigned long flags)
  06.
           /*通过kzalloc分配。*/
  07.
  08.
           buffer->priv_virt = kzalloc(len, GFP_KERNEL);
  09.
           if (!buffer->priv_virt)
  10.
               return -ENOMEM;
  11.
           atomic_add(len, &system_contig_heap_allocated);
                                                                                                 关闭
  12.
           return 0:
其他的几种Heap type可自行研究,接着调用ion_buffer_add将buffer添加到dev的buffers树上去
                   CP
  01.
       static void ion_buffer_add(struct ion_device *dev,
  02.
                      struct ion_buffer *buffer)
  03.
  04.
           struct rb node **p = &dev->buffers.rb node;
           struct rb_node *parent = NULL;
  05.
  06.
           struct ion_buffer *entry;
  07.
```

第62页 共69页 2017年09月07日 04:09

```
08.
            while (*p) {
  09.
  10.
               entry = rb_entry(parent, struct ion_buffer, node);
  11.
               if (buffer < entry) {</pre>
 12.
  13.
                   p = &(*p)->rb_left;
  14.
               } else if (buffer > entry) {
  15.
                   p = &(*p)->rb_right;
  16.
               } else {
  17.
                   pr_err("%s: buffer already found.", __func__);
 18.
                   BUG();
  19.
  20.
  21.
       /*又是使用红黑树哦!*/
           rb_link_node(&buffer->node, parent, p);
  22.
  23.
            rb_insert_color(&buffer->node, &dev->buffers);
  24.
  01.
        static void ion buffer add(struct ion device *dev,
  02.
                      struct ion_buffer *buffer)
  03.
            struct rb_node **p = &dev->buffers.rb_node;
            struct rb_node *parent = NULL;
  05.
           struct ion buffer *entry;
  06.
  07.
  08.
            while (*p) {
  09.
               parent = *p;
  10.
               entry = rb_entry(parent, struct ion_buffer, node);
  11.
               if (buffer < entry) {</pre>
  12.
  13.
                   p = &(*p)->rb_left;
  14.
               } else if (buffer > entry) {
  15.
                   p = &(*p)->rb_right;
  16.
               } else {
  17.
                   pr_err("%s: buffer already found.", __func__);
 18.
                   BUG();
  19.
               }
  20.
  21.
       /*又是使用红黑树哦!*/
  22.
           rb_link_node(&buffer->node, parent, p);
  23.
            rb insert color(&buffer->node, &dev->buffers):
  24.
至此,已经得到client和handle, buffer分配完成!
ION_IOC_MAP/ ION_IOC_SHARE
       int ion_share_dma_buf(struct ion_client *client, struct ion_handle *handle)
  01.
  02.
            struct ion_buffer *buffer;
  03.
  04.
            struct dma_buf *dmabuf;
  05.
            bool valid_handle;
  06.
           int fd;
  07.
           mutex lock(&client->lock);
  08.
  09.
            valid_handle = ion_handle_validate(client, handle);
  10.
            mutex_unlock(&client->lock);
  11.
            if (!valid_handle) {
  12.
               WARN(1, "%s: invalid handle passed to share.\n", __func__);
               return -EINVAL:
 13.
  14.
  15.
                                                                                                   关闭
  16.
            buffer = handle->buffer;
  17.
            ion_buffer_get(buffer);
  18.
            /*生成一个新的file描述符*/
           dmabuf = dma_buf_export(buffer, &dma_buf_ops, buffer->size, O_RDWR);
  19.
  20.
            if (IS_ERR(dmabuf)) {
               ion_buffer_put(buffer);
  21.
  22.
               return PTR_ERR(dmabuf);
  23.
            /*将file转换用户空间识别的fd描述符。*/
  24.
            fd = dma_buf_fd(dmabuf, O_CLOEXEC);
 25.
           if (fd < 0)
  26.
  27.
               dma_buf_put(dmabuf);
  28.
  29.
```

第63页 共69页 2017年09月07日 04:09

```
30.
31.
      struct dma_buf *dma_buf_export(void *priv, const struct dma_buf_ops *ops,
32.
                     size_t size, int flags)
33.
          struct dma_buf *dmabuf;
34.
35.
         struct file *file;
36.
      ~~snip
37.
         /*分配struct dma_buf.*/
38.
         dmabuf = kzalloc(sizeof(struct dma_buf), GFP_KERNEL);
39.
         if (dmabuf == NULL)
40.
             return ERR PTR(-ENOMEM);
         /*保存信息到dmabuf,注意ops为dma_buf_ops,后面mmap为调用到。*/
41.
42.
         dmabuf->priv = priv;
43.
          dmabuf->ops = ops;
          dmabuf->size = size;
45.
          /*产生新的file*/
         file = anon_inode_getfile("dmabuf", &dma_buf_fops, dmabuf, flags);
46.
47.
48.
          dmabuf->file = file;
49.
50.
          mutex_init(&dmabuf->lock);
         INIT_LIST_HEAD(&dmabuf->attachments);
51.
52.
53.
          return dmabuf:
54. }
01.
      int ion_share_dma_buf(struct ion_client *client, struct ion_handle *handle)
02.
          struct ion_buffer *buffer;
03.
          struct dma buf *dmabuf:
04.
         bool valid handle;
05.
06.
          int fd;
07.
          mutex_lock(&client->lock);
          valid_handle = ion_handle_validate(client, handle);
09.
          mutex unlock(&client->lock);
10.
11.
          if (!valid handle) {
12.
              WARN(1, "%s: invalid handle passed to share.\n", __func__);
13.
              return -EINVAL;
14.
15.
         buffer = handle->buffer;
16.
17.
         ion_buffer_get(buffer);
18.
          /*生成一个新的file描述符*/
19.
          dmabuf = dma_buf_export(buffer, &dma_buf_ops, buffer->size, O_RDWR);
         if (IS_ERR(dmabuf)) {
21.
             ion_buffer_put(buffer);
22.
             return PTR ERR(dmabuf);
23.
         /*将file转换用户空间识别的fd描述符。*/
24.
25.
          fd = dma_buf_fd(dmabuf, O_CLOEXEC);
         if (fd < 0)
26.
27.
             dma_buf_put(dmabuf);
28.
29.
         return fd;
30.
31.
     struct dma_buf *dma_buf_export(void *priv, const struct dma_buf_ops *ops,
32.
                     size_t size, int flags)
33.
          struct dma buf *dmabuf:
34.
35.
         struct file *file;
36.
      ~~snip
37.
         /*分配struct dma_buf.*/
                                                                                                 关闭
38.
          dmabuf = kzalloc(sizeof(struct dma_buf), GFP_KERN
         if (dmabuf == NULL)
39.
40.
             return ERR_PTR(-ENOMEM);
          /*保存信息到dmabuf,注意ops为dma_buf_ops,后面mmap为调用到。*/
41.
42.
          dmabuf->priv = priv;
43.
          dmabuf->ops = ops;
          dmabuf->size = size;
45.
          /*产生新的file*/
         file = anon_inode_getfile("dmabuf", &dma_buf_fops, dmabuf, flags);
46.
47.
48.
          dmabuf->file = file;
49.
          mutex_init(&dmabuf->lock);
50.
51.
          INIT_LIST_HEAD(&dmabuf->attachments);
52.
```

第64页 共69页 2017年09月07日 04:09

```
53.
           return dmabuf;
  54. }
通过上述过程,用户空间就得到了新的fd.重新生成一个新的fd的目的是考虑了两个用户空间进程想共享这块heap
内存的情况。然后再对fd作mmap,相应地kernel空间就调用到了file的dma_buf_fops中的
dma_buf_mmap_internal。
       [cpp]
       static const struct file operations dma buf fops = {
  01.
  02.
           .release = dma_buf_release,
  03.
                      = dma_buf_mmap_internal,
  04.
  05.
       static int dma_buf_mmap_internal(struct file *file, struct vm_area_struct *
  06.
  07.
           struct dma buf *dmabuf:
  08.
  09.
           if (!is_dma_buf_file(file))
  10.
               return -EINVAL;
  11.
           dmabuf = file->private_data;
  12.
           /*检查用户空间要映射的size是否比目前dmabuf也就是当前heap的size
 13.
  14.
       还要大,如果是就返回无效。*/
  15.
           /* check for overflowing the buffer's size */
           if (vma->vm_pgoff + ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT) >
  17.
               dmabuf->size >> PAGE_SHIFT)
               return -EINVAL:
 18.
  19.
           /*调用的是dma_buf_ops 的mmap函数*/
  20.
           return dmabuf->ops->mmap(dmabuf, vma);
  21.
        }
  22.
       struct dma_buf_ops dma_buf_ops = {
           .map_dma_buf = ion_map_dma_buf,
  24.
           .unmap dma_buf = ion_unmap_dma_buf,
  25.
  26.
           .mmap = ion mmap,
  27.
           .release = ion_dma_buf_release,
           .begin_cpu_access = ion_dma_buf_begin_cpu_access,
           .end_cpu_access = ion_dma_buf_end_cpu_access,
           .kmap_atomic = ion_dma_buf_kmap,
  30.
  31.
           .kunmap_atomic = ion_dma_buf_kunmap,
  32.
           .kmap = ion dma buf kmap,
  33.
           .kunmap = ion_dma_buf_kunmap,
  34.
       static int ion_mmap(struct dma_buf *dmabuf, struct vm_area_struct *vma)
  36.
           struct ion_buffer *buffer = dmabuf->priv;
  37.
  38.
           int ret;
  39.
           if (!buffer->heap->ops->map_user) {
  40.
  41.
               pr_err("%s: this heap does not define a method for mapping "
                      "to userspace\n", __func__);
  42.
               return -EINVAL:
  43.
  44.
  45.
  46.
           mutex_lock(&buffer->lock);
           /* now map it to userspace */
  48.
           /*调用的是相应heap的map_user,如carveout_heap_ops 调用的是
       ion_carveout_heap_map_user ,此函数就是一般的mmap实现,不追下去了。*/
  49.
  50.
           ret = buffer->heap->ops->map_user(buffer->heap, buffer, vma);
  51.
  53.
               mutex_unlock(&buffer->lock);
               pr_err("%s: failure mapping buffer to userspace\n",
  54.
  55.
                     __func__);
                                                                                                关闭
  56.
           } else {
  57.
               buffer->umap_cnt++;
               mutex_unlock(&buffer->lock);
  59.
  60.
               vma->vm_ops = &ion_vm_ops;
  61.
                ^{\star} move the buffer into the <code>vm_private_data</code> so we can access it
  62.
                * from vma_open/close
  63.
  65.
               vma->vm_private_data = buffer;
  66.
  67.
           return ret:
  68. }
```

第65页 共69页 2017年09月07日 04:09

```
[cpp]
  01.
       static const struct file operations dma buf fons = {
  02.
           .release = dma buf release,
  03.
                     = dma_buf_mmap_internal,
  04.
       static int dma_buf_mmap_internal(struct file *file, struct vm_area_struct *vma)
  05.
  06.
           struct dma buf *dmabuf;
  07.
  08.
  09.
           if (!is_dma_buf_file(file))
  10.
               return -EINVAL;
  11.
          dmabuf = file->private_data;
  12.
           /*检查用户空间要映射的size是否比目前dmabuf也就是当前heap的size
  13.
       还要大,如果是就返回无效。*/
 14.
  15.
           /* check for overflowing the buffer's size */
  16.
           if (vma->vm_pgoff + ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT) >
               dmabuf->size >> PAGE_SHIFT)
  18.
               return -EINVAL:
           /*调用的是dma_buf_ops 的mmap函数*/
  19.
  20.
           return dmabuf->ops->mmap(dmabuf, vma);
  21.
  22.
  23.
       struct dma_buf_ops dma_buf_ops = {
          .map_dma_buf = ion_map_dma_buf,
  24.
  25.
           .unmap_dma_buf = ion_unmap_dma_buf,
  26.
           .mmap = ion mmap,
  27.
           .release = ion_dma_buf_release,
  28.
           .begin_cpu_access = ion_dma_buf_begin_cpu_access,
           .end_cpu_access = ion_dma_buf_end_cpu_access,
  30.
           .kmap_atomic = ion_dma_buf_kmap,
           .kunmap_atomic = ion_dma_buf_kunmap,
  31.
  32.
           .kmap = ion dma buf kmap,
  33.
           .kunmap = ion_dma_buf_kunmap,
  34.
  35.
       static int ion_mmap(struct dma_buf *dmabuf, struct vm_area_struct *vma)
  36.
  37.
           struct ion buffer *buffer = dmabuf->priv:
  38.
           int ret;
  39.
           if (!buffer->heap->ops->map_user) {
  40.
               pr_err("%s: this heap does not define a method for mapping "
  42.
                      "to userspace\n", __func__);
               return -EINVAL:
  43.
  44.
  45.
  46.
           mutex_lock(&buffer->lock);
  47.
           /* now map it to userspace */
  48.
           /*调用的是相应heap的map_user,如carveout_heap_ops 调用的是
       ion_carveout_heap_map_user ,此函数就是一般的mmap实现,不追下去了。*/
  49.
  50.
           ret = buffer->heap->ops->map_user(buffer->heap, buffer, vma);
  51.
  52.
           if (ret) {
  53.
              mutex_unlock(&buffer->lock);
               pr_err("%s: failure mapping buffer to userspace\n",
  54.
  55.
                      __func__);
  56.
           } else {
  57.
               buffer->umap_cnt++;
               mutex_unlock(&buffer->lock);
  59.
  60.
               vma->vm_ops = &ion_vm_ops;
  61.
                ^{\star} move the buffer into the <code>vm_private_data</code> so we can access it
  62.
                * from vma_open/close
  63.
  64.
                                                                                                关闭
               vma->vm_private_data = buffer;
  66.
  67.
           return ret;
至此,用户空间就得到了bufferaddress,然后可以使用了!
ION_IOC_IMPORT
当用户空间另一个进程需要这块heap的时候,ION_IOC_IMPORT就派上用处了!注意,
传进去的fd为在ION_IOC_SHARE中得到的。
                 C Y
       [cpp]
```

第66页 共69页 2017年09月07日 04:09

```
01.
       struct ion handle *ion import dma buf(struct ion client *client, int fd)
  02.
  03.
  04.
           struct dma_buf *dmabuf;
           struct ion buffer *buffer:
  05.
  06.
           struct ion_handle *handle;
  07.
           dmabuf = dma_buf_get(fd);
  08.
  09.
           if (IS_ERR_OR_NULL(dmabuf))
  10.
              return ERR_PTR(PTR_ERR(dmabuf));
           /* if this memory came from ion */
  11.
  12.
       ~~snip
  13.
           buffer = dmabuf->priv;
  14.
  15.
           mutex_lock(&client->lock);
  16.
           /* if a handle exists for this buffer just take a reference to it */
       /*查找是否已经存在对应的handle了,没有则创建。因为另外一个进程只是
 17.
  18.
       调用了open 接口,对应的只创建了client,并没有handle。
  19.
  20.
           handle = ion_handle_lookup(client, buffer);
  21.
           if (!IS_ERR_OR_NULL(handle)) {
 22.
              ion_handle_get(handle);
  23.
               qoto end:
  24.
  25.
           handle = ion handle create(client, buffer);
  26.
           if (IS_ERR_OR_NULL(handle))
               goto end;
  28.
           ion handle add(client, handle);
  29.
  30.
           mutex_unlock(&client->lock);
  31.
           dma_buf_put(dmabuf);
  32.
           return handle;
  33.
       struct ion_handle *ion_import_dma_buf(struct ion_client *client, int fd)
  01.
  02.
  03.
  04.
           struct dma_buf *dmabuf;
  05.
           struct ion_buffer *buffer;
           struct ion_handle *handle;
  07.
           dmabuf = dma buf get(fd);
  08.
  09.
           if (IS_ERR_OR_NULL(dmabuf))
  10.
               return ERR_PTR(PTR_ERR(dmabuf));
  11.
           /* if this memory came from ion *
  12.
       ~~snip
  13.
           buffer = dmabuf->priv;
  14.
  15.
           mutex_lock(&client->lock);
  16.
           ^{\prime *} if a handle exists for this buffer just take a reference to it ^{*}/
  17.
       /*查找是否已经存在对应的handle了,没有则创建。因为另外一个进程只是
       调用了open 接口,对应的只创建了client,并没有handle。
  18.
  19.
  20.
           handle = ion_handle_lookup(client, buffer);
  21.
           if (!IS ERR OR NULL(handle)) {
  22.
               ion_handle_get(handle);
  23.
  25.
           handle = ion_handle_create(client, buffer);
  26.
           if (IS_ERR_OR_NULL(handle))
  27.
               qoto end;
  28.
           ion_handle_add(client, handle);
  29.
  30.
                                                                                                关闭
           mutex_unlock(&client->lock);
  31.
           dma_buf_put(dmabuf);
  32.
           return handle:
  33.
这样,用户空间另一个进程也得到了对应的bufferHandle,client/buffer/handle之间连接起来了!然后另一个一个
进程就也可以使用mman来操作这块hean buffer了。
```

和一般的进程使用ION区别就是共享的进程之间struction\_buffer是共享的,而struct ion\_handle是各自的。可见,ION的使用流程还是比较清晰的。不过要记得的是,使用好了ION,一定要释放掉,否则会导致内存泄露。

## ION内核空间使用

内核空间使用ION也是大同小异,按照创建client,buffer,handle的流程,只是它的使用对用户空间来说是透明的罢

第67页 共69页 2017年09月07日 04:09

```
了!
 ion client create在kernel空间被Qualcomm给封装了下。
               CY
       struct ion_client *msm_ion_client_create(unsigned int heap_mask,
   02.
                       const char *name)
   03.
   04.
           return ion_client_create(idev, heap_mask, name);
   05. }
   01.
       struct ion_client *msm_ion_client_create(unsigned int heap_mask,
                       const char *name)
   03.
           return ion_client_create(idev, heap_mask, name);
   04.
   05. }
 调用的流程也类似,不过map的时候调用的是heap对应的map_kernel()而不是map_user().
 msm_ion_client_create -> ion_alloc ->ion_map_kernel
 参考文档:
 http://lwn.net/Articles/480055/
  《ARM体系结构与编程》存储系统章节。
                                 顶
    上一篇 Camera服务之--Service
    下一篇 Android学习之ION memory manager
  相关文章推荐
   • ION基本概念介绍和原理分析
                                         • 嵌入式资料---嵌入式基本概念原理和入门知识
   • 轻松拿下Linux进程、线程和调度
                                         · 一招学会Android自定义控件

    Bayesian Network 基本概念和原理

                                          • gprs基本原理GPRS基本概念问答
   • 30天掌握机器学习升级版

    从零练就iOS高手

   • 2016计算机组成原理研究生入学统考基本概念及...
                                         • RF射频知识基本概念及DTD无线产品介绍
   • Python网络爬虫快速入门实战
                                          • ION基本概念介绍
   • 深入研究Windows内部原理系列之四:Windows...

    ATM技术 基本概念 协议原理 业务类型

   • 最适合自学的C++基础知识
                                          • SAP:销售与分销(介绍SAP的SD模块的基本概念...
查看评论
                                                                                  关闭
暂无评论
*以上用户言论只代表其个人观点,不代表CSDN网站的观点或立场
```

第68页 共69页 2017年09月07日 04:09

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved



关闭

第69页 共69页 2017年09月07日 04:09