# 一、概述

如图所示，在海思平台上将内存分为两个部分：os内存和mmz内存。os内存指：由linux操作系统管理的内存；mmz内存：由mmz驱动模块进行管理供媒体业务单独使用的内存，在驱动加载时可以指定该模块管理内存的大小：

insmod mmz.ko mmz=anonymous,0,0x4fa00000,6M anony=1 || report\_error



该驱动主要由两个文件组成：media-mem.c和mmz-userdev.c，加载驱动后相应的设备文件：/dev/mmz\_userdev，应用层通过打开该设备文件进行ioctl（申请mmz内存、释放mmz内存、重映射mmz内存到内核等）和直接mmap操作，而媒体底层驱动模块则直接调用mmz驱动的导出接口进行相应操作。

# 二、数据结构

## 1、mmz区域描述符

hil\_media\_memory\_zone描述了一个mmz区域的所有信息，可以有多个mmz区域，通过链表连接在一起。

struct hil\_media\_memory\_zone {

char name[HIL\_MMZ\_NAME\_LEN+1]; //mmz区域名字：anonymous

unsigned long gfp; //区域标识：0

unsigned long phys\_start; //mmz区域起始物理地址：0x4fa00000

unsigned long nbytes; //mmz区域大小：6M

struct list\_head list; //mmz链表

unsigned char \*bitmap; //位图

struct list\_head mmb\_list; //mmz区域的mmb链表，存放所有申请到的物理内存

unsigned int alloc\_type;

unsigned long block\_align;

void (\*destructor)(const void \*);

};

## 2、mmb内存描述符

hil\_media\_memory\_block描述了从mmz区域申请一块内存，同一个mmz区域内的所有mmb通过链表连接。

struct hil\_media\_memory\_block {

#ifndef MMZ\_V2\_SUPPORT

unsigned int id;

#endif

char name[HIL\_MMB\_NAME\_LEN+1]; //该mmb模块使用者名字

struct hil\_media\_memory\_zone \*zone; //指向mmb所属的mmz区域

struct list\_head list; //mmb链表

unsigned long phys\_addr; //申请到的mmb起始物理地址

void \*kvirt; //对应内核虚拟地址，从代码看未用

unsigned long length; //申请的mmb大小

unsigned long flags; //标识

unsigned int order;

int phy\_ref; //引用计数

int map\_ref; //引用计数

};

3、mmz\_userdev\_info

该结构体保存打开该设备文件的进程信息，存放在file结构体的private\_data成员里。

struct mmz\_userdev\_info {

pid\_t pid; //打开设备文件的进程pid

pid\_t mmap\_pid;

struct semaphore sem; //信号量

struct list\_head list; //指向mmb\_info链表

};

4、mmb\_info

该结构体描述应用申请到mmb后的相关信息，同进程的mmb\_info通过链表形式管理。

struct mmb\_info {

unsigned long phys\_addr; //申请到的物理内存，同mmb.phys.addr

unsigned long align; /\* if you need your phys-memory have special align size \*/

unsigned long size; //申请的物理内存大小

unsigned int order;

void \*mapped; //指向mmap后的虚拟地址空间

union {

struct {

unsigned long prot :8; /\* PROT\_READ or PROT\_WRITE \*/

unsigned long flags :12;/\* MAP\_SHARED or MAP\_PRIVATE \*/

#ifdef \_\_KERNEL\_\_

unsigned long reserved :8; /\* reserved, do not use \*/

unsigned long delayed\_free :1;

unsigned long map\_cached :1;

#endif

};

unsigned long w32\_stuf;

};

char mmb\_name[HIL\_MMB\_NAME\_LEN+1];

char mmz\_name[HIL\_MMZ\_NAME\_LEN+1];

unsigned long gfp; /\* reserved, do set to 0 \*/

#ifdef \_\_KERNEL\_\_

int map\_ref;

int mmb\_ref;

struct list\_head list; //mmb\_info链表

hil\_mmb\_t \*mmb; //指向申请到的mmb

#endif

};

# 三、关系图

## 1、mmz和mmb关系

下图展示了mmz驱动管理mmz和mmb的关系。mmz驱动模块支持多个mmz区域，只要在加载mmz.ko时通过参数传递即可，一般情况下只有一个mmz区域。多个mmz区域之间通过链表的形式组织在一起，链表头为mmz\_list；而每个mmz区域通过mmb\_list维护mmb链表，管理该区域内所有已经申请了的物理内存区域；每个mmb通过zone成员知道自己属于哪个mmz区域。

通过/proc/media-mem可以查看mmz和mmb使用情况：

+---ZONE: PHYS(0x4FA00000, 0x4FFFFFFF), GFP=0, nBYTES=6144KB, NAME="anonymous"

|-MMB: phys(0x4FA00000, 0x4FA81FFF), kvirt=0x (null), flags=0x00000000, length=520KB, name="DCCM\_MSG\_BUF"

|-MMB: phys(0x4FA82000, 0x4FA84FFF), kvirt=0x (null), flags=0x00000000, length=12KB, name="SYS\_scale\_coef"

|-MMB: phys(0x4FA85000, 0x4FA87FFF), kvirt=0x (null), flags=0x00000000, length=12KB, name="SYS\_scale\_coef"

|-MMB: phys(0x4FA88000, 0x4FB07FFF), kvirt=0x (null), flags=0x00000000, length=512KB, name="TDE\_MemPool"

|-MMB: phys(0x4FB08000, 0x4FB3CFFF), kvirt=0x (null), flags=0x00000000, length=212KB, name="IVE\_QUEUE"

|-MMB: phys(0x4FB3D000, 0x4FB3DFFF), kvirt=0x (null), flags=0x00000000, length=4KB, name="IVE\_TEMP\_NODE"

---MMZ\_USE\_INFO:

total size=6144KB(6MB),used=1272KB(1MB + 248KB),remain=4872KB(4MB + 776KB),zone\_number=1,block\_number=6



## 2、mmb、mmb\_info和mmb\_userdev\_info关系

下图展示了mmz驱动和应用申请数据结构之间的关系：



当应用打开设备文件/dev/mmz\_userdev时会申请一个属于该进程的mmb\_userdev\_info结构体，mmb\_userdev\_info成员list指向属于该进程的所有mmb\_info，mmb\_info的mmb成员指向为其分配的mmb，而\*mmaped存放mmb物理内存（phy\_addr）进行映射后的虚拟地址供用户空间使用。

由于mmz大部分为媒体业务独立使用，内存在媒体硬件模块流转，应用无需访问，这时不用映射，只有当应用需要访问时才需要进行映射。可以把mmz管理的整个内存看做存储盘上的一个大文件，应用层要访问mmz的物理内存通过mmap方式进行映射（类似文件），映射的文件偏移就是mmb.phy\_addr，映射大小就是申请的mmb.length，通过mmap方式映射到内核后根据返回的虚拟地址就可以访问该mmb内存。