# Linux驱动\_mmap

## 1. 相关知识

当内核空间和用户空间存在大量数据交互时，共享内存映射就成了这种情况下的不二选择，它能够最大限度的降低内核空间和用户空间之间的数据拷贝，从而大大提高系统的性能。

### 1.1 Linux内核常用的内存空间申请方式

内核空间申请内存主要函数有：kmalloc()，\_\_get\_free\_pages()他们申请的内存位于物理内存映射区，且在物理上是连续的，与真实的物理地址只有一个固定的偏移。vmalloc()申请的连续虚拟内存空间在物理上则不一定连续，它们之间也没有简单的换算关系。

### 1.2 mmap基本概念

mmap()实现了将用户空间的一段内存与设备内存关联，当用户访问用户空间的这段地址范围时，会转化为对设备的访问。mmap()必须以PAGE\_SIZE为单位进行映射。

调用mmap()的时候，内核会进行如下处理：

(1)在进程的虚拟空间查找一块VMA

(2)将这块VMA进行映射

(3)如果设备驱动程序或文件系统的file\_operations定义了mmap()操作，则调用它

(4)将这个VMA插入到进程的VMA链表中

驱动程序中的mmap()实现机制是建立页表，并填充VMA结构体中vm\_operations\_struct指针。针对VMA的操作都被包含在vm\_operations\_struct结构体中。

### 1.3 写缓存和Cache

通常,I/O内存被映射时需要是nocache的，这时我们需要对vma->vm\_page\_prot设置nocache标志之后再映射。

vma**->**vm\_page\_prot **=** pgprot\_noncached**(**vma**->**vm\_page\_prot**);**//赋nocache标志。

pgprot\_noncached()禁止了相关页的Cache和写缓冲（write buffer），pgprot\_writecombine()则没有禁止写缓冲。ARM下的写缓冲器是一个非常小的FIFO存储器，位于处理器核与主存之间，其目的将处理器核和Cache从较慢的主存写操作中解脱出来。

在LCD驱动中，framebuffer物理内存空间采用dma\_alloc\_writecombine函数分配出来。dma\_alloc\_coherent 在 arm 平台上会禁止页表项中的 C（Cacheable域以及B (Bufferable)域。而dma\_alloc\_writecombine只禁止C（Cacheable）域。

## 2. mmap驱动

mmap映射的方法：

有两种方法建立页表，

(1)一次性建立页表，可以调用函数remap\_pfn\_range

(2)每次建立一个页的页表，调用函数nopage。

### 2.1 remap\_pfn\_range方法

利用vmalloc或kmalloc申请内存控件，调用remap\_pfn\_range将内存空间映射到用户空间。

#### 2.1.1 驱动代码

#include <linux/module.h>

#include <linux/types.h>

#include <linux/mm.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/device.h>

#include <linux/io.h>

#include <linux/cdev.h> //cdev\_init等函数定义

#include <asm/uaccess.h>

#define USE\_KMALLOC 0

#define MEMC\_SIZE 4096

#define DEVICE\_NAME "memc"

int memc\_major**;**

char **\***memc\_data**;**

static struct class **\***memcdrv\_class**;**

struct memc\_dev\_t

**{**

struct cdev cdev**;**

**}**memc\_dev**;**

static int memc\_mmap**(**struct file**\***filp**,** struct vm\_area\_struct **\***vma**)**

**{**

#if !USE\_KMALLOC

unsigned long pfn**;**

int ret**;**

unsigned long start **=** vma**->**vm\_start**;**

#endif

unsigned long size **=** PAGE\_ALIGN**(**vma**->**vm\_end **-** vma**->**vm\_start**);**

printk**(**"mmap size=%ld\n"**,**size**);**

**if(**size **>** MEMC\_SIZE**)**

**{**

**return** **-**EINVAL**;**

**}**

vma**->**vm\_flags **|=** VM\_IO**;**

vma**->**vm\_flags **|=** VM\_RESERVED**;**

#if USE\_KMALLOC

**if** **(**remap\_pfn\_range**(**vma**,**vma**->**vm\_start**,**virt\_to\_phys**(**memc\_data**)>>**PAGE\_SHIFT**,** size**,** vma**->**vm\_page\_prot**))**

**return** **-**EAGAIN**;**

#else

**while** **(**size **>** 0**)**

**{**

pfn **=** vmalloc\_to\_pfn**(**memc\_data**);**

**if** **((**ret **=** remap\_pfn\_range**(**vma**,** start**,** pfn**,** PAGE\_SIZE**,** vma**->**vm\_page\_prot**))** **<** 0**)**

**{**

**return** ret**;**

**}**

start **+=** PAGE\_SIZE**;**

memc\_data **+=** PAGE\_SIZE**;**

size **-=** PAGE\_SIZE**;**

**}**

#endif

**return** 0**;**

**}**

static int memc\_open**(**struct inode **\***inode**,** struct file **\***filp**)**

**{**

**return** 0**;**

**}**

static int memc\_release**(**struct inode **\***inode**,** struct file **\***filp**)**

**{**

**return** 0**;**

**}**

static const struct file\_operations memc\_fops **=**

**{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** memc\_open**,**

**.**release **=** memc\_release**,**

**.**mmap **=** memc\_mmap**,**

**};**

static int \_\_init memc\_init**(**void**)**

**{**

dev\_t dev\_id**;**

**if(**memc\_major**)**

**{**

dev\_id **=** MKDEV**(**memc\_major**,**0**);**

register\_chrdev\_region**(**dev\_id**,**1**,**DEVICE\_NAME**);**

**}**

**else**

**{**

//动态

alloc\_chrdev\_region**(&**dev\_id**,**0**,**1**,**DEVICE\_NAME**);**

memc\_major **=** MAJOR**(**dev\_id**);**

**}**

cdev\_init**(&**memc\_dev**.**cdev**,** **&**memc\_fops**);**

memc\_dev**.**cdev**.**owner **=** THIS\_MODULE**;**

cdev\_add**(&**memc\_dev**.**cdev**,**dev\_id**,**1**);**

memcdrv\_class **=** class\_create**(**THIS\_MODULE**,** DEVICE\_NAME**);**

class\_device\_create**(**memcdrv\_class**,** **NULL,** MKDEV**(**memc\_major**,** 0**),** **NULL,** DEVICE\_NAME**);**

#if USE\_KMALLOC

memc\_data **=** kzalloc**(**MEMC\_SIZE**,** GFP\_KERNEL**);**

#else

memc\_data **=** vmalloc**(**MEMC\_SIZE**);**

memset**(**memc\_data**,**0x00**,**MEMC\_SIZE**);**

#endif

**return** 0**;**

**}**

static void \_\_exit memc\_exit**()**

**{**

class\_device\_destroy**(**memcdrv\_class**,**MKDEV**(**memc\_major**,**0**));**

class\_destroy**(**memcdrv\_class**);**

cdev\_del**(&**memc\_dev**.**cdev**);**

unregister\_chrdev\_region**(**MKDEV**(**memc\_major**,**0**),**1**);**

#if USE\_KMALLOC

kfree**(**memc\_data**);**

#else

vfree**(**memc\_data**);**

#endif

**}**

module\_init**(**memc\_init**);**

module\_exit**(**memc\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

这段驱动代码有个缺陷，没有考虑mmap映射时的偏移地址，假定了偏移地址为0。在mmap时offset必须是分页大小的整数倍。

#### 2.1.2 测试代码

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

int main**()**

**{**

int fd**;**

char **\***start**;**

//char buf[100];

char **\***buf**;**

/\*打开文件\*/

fd **=** open**(**"/dev/memc"**,**O\_RDWR**);**

buf **=** **(**char **\*)**malloc**(**100**);**

memset**(**buf**,** 0**,** 100**);**

start **=** mmap**(NULL,**100**,**PROT\_READ**|**PROT\_WRITE**,**MAP\_SHARED**,**fd**,**0**);**

/\* 读出数据 \*/

strcpy**(**buf**,**start**);**

sleep **(**1**);**

printf**(**"buf 1 = %s\n"**,**buf**);**

/\* 写入数据 \*/

strcpy**(**start**,**"Buf Is Not Null!"**);**

memset**(**buf**,** 0**,** 100**);**

strcpy**(**buf**,**start**);**

sleep **(**1**);**

printf**(**"buf 2 = %s\n"**,**buf**);**

munmap**(**start**,**100**);** /\*解除映射\*/

free**(**buf**);**

close**(**fd**);**

**return** 0**;**

**}**

#### 2.1.3 测试结果

# insmod mmap\_drv.ko

# ./mmap\_drv\_test

mmap size**=**4096

buf 1 **=**

buf 2 **=** Buf Is Not Null**!**

### 2.2 nopage方法

nopage方法借助缺页异常，当访问不存在的页时，会寻求映射。

#### 2.2.1 驱动代码

#include <linux/module.h>

#include <linux/types.h>

#include <linux/mm.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/device.h>

#include <linux/io.h>

#include <linux/cdev.h> //cdev\_init等函数定义

#include <asm/uaccess.h>

#include <linux/slab.h>

#define USE\_KMALLOC 1

#define MEMC\_SIZE 4096

#define DEVICE\_NAME "memc"

int memc\_major**;**

char **\***memc\_data**;**

static struct class **\***memcdrv\_class**;**

struct memc\_dev\_t

**{**

struct cdev cdev**;**

**}**memc\_dev**;**

static struct page**\*** nopage\_fault**(**struct vm\_area\_struct **\***vma**,**unsigned long address**,** int **\***type**)**

**{**

//每次产生一页的映射

printk**(**"nopage\_fault\n"**);**

struct page **\***page**;**

//address： 缺页时的虚拟地址

//vm\_start：映射的起始虚拟地址

//vm\_pgoff: 映射时的偏移

unsigned long offset **=** address **-** vma**->**vm\_start **+** vma**->**vm\_pgoff**;**

**if(**offset **>** MEMC\_SIZE**)**

**return** NOPAGE\_SIGBUS**;**

//对应的memc\_data虚拟地址

unsigned long virtaddr **=** memc\_data **+** offset**;**

//这里便得到了缺页的实际物理地址对应的page

#if USE\_KMALLOC

page **=** virt\_to\_page**(**virtaddr**);**

#else

page **=** vmalloc\_to\_page**(**virtaddr**);**

#endif

get\_page**(**page**);**//增加该页的使用计数

**return** page**;**

**}**

static struct vm\_operations\_struct nopage\_vm\_ops **=**

**{**

**.**nopage **=** nopage\_fault**,**

**};**

static int memc\_mmap**(**struct file**\***filp**,** struct vm\_area\_struct **\***vma**)**

**{**

vma**->**vm\_flags **|=** VM\_RESERVED**;**

vma**->**vm\_ops **=** **&**nopage\_vm\_ops**;**

**return** 0**;**

**}**

static int memc\_open**(**struct inode **\***inode**,** struct file **\***filp**)**

**{**

**return** 0**;**

**}**

static int memc\_release**(**struct inode **\***inode**,** struct file **\***filp**)**

**{**

**return** 0**;**

**}**

static const struct file\_operations memc\_fops **=**

**{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** memc\_open**,**

**.**release **=** memc\_release**,**

**.**mmap **=** memc\_mmap**,**

**};**

static int \_\_init memc\_init**(**void**)**

**{**

dev\_t dev\_id**;**

**if(**memc\_major**)**

**{**

dev\_id **=** MKDEV**(**memc\_major**,**0**);**

register\_chrdev\_region**(**dev\_id**,**1**,**DEVICE\_NAME**);**

**}**

**else**

**{**

//动态

alloc\_chrdev\_region**(&**dev\_id**,**0**,**1**,**DEVICE\_NAME**);**

memc\_major **=** MAJOR**(**dev\_id**);**

**}**

cdev\_init**(&**memc\_dev**.**cdev**,** **&**memc\_fops**);**

memc\_dev**.**cdev**.**owner **=** THIS\_MODULE**;**

cdev\_add**(&**memc\_dev**.**cdev**,**dev\_id**,**1**);**

memcdrv\_class **=** class\_create**(**THIS\_MODULE**,** DEVICE\_NAME**);**

class\_device\_create**(**memcdrv\_class**,** **NULL,** MKDEV**(**memc\_major**,** 0**),** **NULL,** DEVICE\_NAME**);**

#if USE\_KMALLOC

memc\_data **=** kzalloc**(**MEMC\_SIZE**,** GFP\_KERNEL**);**

#else

memc\_data **=** vmalloc**(**MEMC\_SIZE**);**

memset**(**memc\_data**,**0x00**,**MEMC\_SIZE**);**

#endif

**return** 0**;**

**}**

static void \_\_exit memc\_exit**()**

**{**

class\_device\_destroy**(**memcdrv\_class**,**MKDEV**(**memc\_major**,**0**));**

class\_destroy**(**memcdrv\_class**);**

cdev\_del**(&**memc\_dev**.**cdev**);**

unregister\_chrdev\_region**(**MKDEV**(**memc\_major**,**0**),**1**);**

#if USE\_KMALLOC

kfree**(**memc\_data**);**

#else

vfree**(**memc\_data**);**

#endif

**}**

module\_init**(**memc\_init**);**

module\_exit**(**memc\_exit**);**

MODULE\_LICENSE**(**"GPL"**);**

测试代码和测试结果和remap\_pfn\_range方法一致。