<http://blog.csdn.net/dlutbrucezhang/article/details/9080173>

[Linux 内存映射函数 mmap（）函数详解](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

分类： [Linux System Programming](http://blog.csdn.net/DLUTBruceZhang/article/category/1442457)2013-06-12 19:02 924人阅读 [评论](http://blog.csdn.net/dlutbrucezhang/article/details/9080173#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/dlutbrucezhang/article/details/9080173#report)

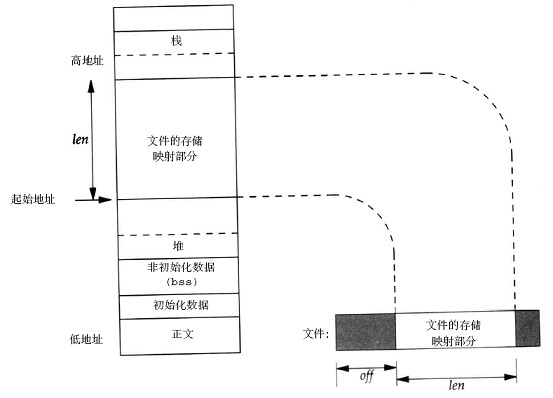
[Linux](http://blog.csdn.net/tag/details.html?tag=Linux)[内存](http://blog.csdn.net/tag/details.html?tag=%e5%86%85%e5%ad%98)[地图](http://blog.csdn.net/tag/details.html?tag=%e5%9c%b0%e5%9b%be)[内核](http://blog.csdn.net/tag/details.html?tag=%e5%86%85%e6%a0%b8)[行业数据](http://blog.csdn.net/tag/details.html?tag=%e8%a1%8c%e4%b8%9a%e6%95%b0%e6%8d%ae)

**一、概述**

**内存映射，简而言之就是将用户空间的一段内存区域映射到内核空间，映射成功后，用户对这段内存区域的修改可以直接反映到内核空间，同样，内核空间对这段区域的修改也直接反映用户空间。那么对于内核空间<---->用户空间两者之间需要大量数据传输等操作的话效率是非常高的。**

**以下是一个把普遍文件映射到用户空间的内存区域的示意图。**

**图一：**



**二、基本函数**

**mmap函数是unix/linux下的系统调用，详细内容可参考《Unix Netword programming》卷二12.2节。**

**mmap系统调用并不是完全为了用于共享内存而设计的。它本身提供了不同于一般对普通文件的访问方式，进程可以像读写内存一样对普通文件的操作。而Posix或系统V的共享内存IPC则纯粹用于共享目的，当然mmap()实现共享内存也是其主要应用之一。**

**mmap系统调用使得进程之间通过映射同一个普通文件实现共享内存。普通文件被映射到进程地址空间后，进程可以像访问普通内存一样对文件进行访问，不必再调用read()，write（）等操作。mmap并不分配空间, 只是将文件映射到调用进程的地址空间里（但是会占掉你的 virutal memory）, 然后你就可以用memcpy等操作写文件, 而不用write()了.写完后，内存中的内容并不会立即更新到文件中，而是有一段时间的延迟，你可以调用msync()来显式同步一下, 这样你所写的内容就能立即保存到文件里了.这点应该和驱动相关。 不过通过mmap来写文件这种方式没办法增加文件的长度, 因为要映射的长度在调用mmap()的时候就决定了.如果想取消内存映射，可以调用munmap()来取消内存映射**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. <span style="background-color: rgb(255, 255, 255);">**void** \* mmap(**void** \*start, **size\_t** length, **int** prot , **int** flags, **int** fd, off\_t offset)</span>

**mmap用于把文件映射到内存空间中，简单说mmap就是把一个文件的内容在内存里面做一个映像。映射成功后，用户对这段内存区域的修改可以直接反映到内核空间，同样，内核空间对这段区域的修改也直接反映用户空间。那么对于内核空间<---->用户空间两者之间需要大量数据传输等操作的话效率是非常高的。**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. start：要映射到的内存区域的起始地址，通常都是用NULL（NULL即为0）。NULL表示由内核来指定该内存地址

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. length：要映射的内存区域的大小

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. prot：期望的内存保护标志，不能与文件的打开模式冲突。是以下的某个值，可以通过or运算合理地组合在一起
2. PROT\_EXEC //页内容可以被执行
3. PROT\_READ  //页内容可以被读取
4. PROT\_WRITE //页可以被写入
5. PROT\_NONE  //页不可访问

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. flags：指定映射对象的类型，映射选项和映射页是否可以共享。它的值可以是一个或者多个以下位的组合体
2. MAP\_FIXED ：使用指定的映射起始地址，如果由start和len参数指定的内存区重叠于现存的映射空间，重叠部分将会被丢弃。如果指定的起始地址不可用，操作将会失败。并且起始地址必须落在页的边界上。
3. MAP\_SHARED ：对映射区域的写入数据会复制回文件内, 而且允许其他映射该文件的进程共享。
4. MAP\_PRIVATE ：建立一个写入时拷贝的私有映射。内存区域的写入不会影响到原文件。这个标志和以上标志是互斥的，只能使用其中一个。
5. MAP\_DENYWRITE ：这个标志被忽略。
6. MAP\_EXECUTABLE ：同上
7. MAP\_NORESERVE ：不要为这个映射保留交换空间。当交换空间被保留，对映射区修改的可能会得到保证。当交换空间不被保留，同时内存不足，对映射区的修改会引起段违例信号。
8. MAP\_LOCKED ：锁定映射区的页面，从而防止页面被交换出内存。
9. MAP\_GROWSDOWN ：用于堆栈，告诉内核VM系统，映射区可以向下扩展。
10. MAP\_ANONYMOUS ：匿名映射，映射区不与任何文件关联。
11. MAP\_ANON ：MAP\_ANONYMOUS的别称，不再被使用。
12. MAP\_FILE ：兼容标志，被忽略。
13. MAP\_32BIT ：将映射区放在进程地址空间的低2GB，MAP\_FIXED指定时会被忽略。当前这个标志只在x86-64平台上得到支持。
14. MAP\_POPULATE ：为文件映射通过预读的方式准备好页表。随后对映射区的访问不会被页违例阻塞。
15. MAP\_NONBLOCK ：仅和MAP\_POPULATE一起使用时才有意义。不执行预读，只为已存在于内存中的页面建立页表入口。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. fd：文件描述符（由open函数返回）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. offset：表示被映射对象（即文件）从那里开始对映，通常都是用0。 该值应该为大小为PAGE\_SIZE的整数倍

**返回说明**

**成功执行时，mmap()返回被映射区的指针，munmap()返回0。失败时，mmap()返回MAP\_FAILED[其值为(void \*)-1]，munmap返回-1。errno被设为以下的某个值**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. EACCES：访问出错
2. EAGAIN：文件已被锁定，或者太多的内存已被锁定
3. EBADF：fd不是有效的文件描述词
4. EINVAL：一个或者多个参数无效
5. ENFILE：已达到系统对打开文件的限制
6. ENODEV：指定文件所在的文件系统不支持内存映射
7. ENOMEM：内存不足，或者进程已超出最大内存映射数量
8. EPERM：权能不足，操作不允许
9. ETXTBSY：已写的方式打开文件，同时指定MAP\_DENYWRITE标志
10. SIGSEGV：试着向只读区写入
11. SIGBUS：试着访问不属于进程的内存区

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. **int** munmap(**void** \*start, **size\_t** length)

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. start：要取消映射的内存区域的起始地址

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. length：要取消映射的内存区域的大小。

**返回说明**

**成功执行时munmap()返回0。失败时munmap返回-1.**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. **int** msync(**const** **void** \*start, **size\_t** length, **int** flags);

**对映射内存的内容的更改并不会立即更新到文件中，而是有一段时间的延迟，你可以调用msync()来显式同步一下, 这样你内存的更新就能立即保存到文件里**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. start：要进行同步的映射的内存区域的起始地址。
2. length：要同步的内存区域的大小
3. flag:flags可以为以下三个值之一：
4. MS\_ASYNC : 请Kernel快将资料写入。
5. MS\_SYNC : 在msync结束返回前，将资料写入。
6. MS\_INVALIDATE : 让核心自行决定是否写入，仅在特殊状况下使用

**三、用户空间和驱动程序的内存映射**

**3.1、基本过程**

**首先，驱动程序先分配好一段内存，接着用户进程通过库函数mmap()来告诉内核要将多大的内存映射到内核空间，内核经过一系列函数调用后调用对应的驱动程序的file\_operation中指定的mmap函数，在该函数中调用remap\_pfn\_range()来建立映射关系。**

**3.2、映射的实现**

**首先在驱动程序分配一页大小的内存，然后用户进程通过mmap()将用户空间中大小也为一页的内存映射到内核空间这页内存上。映射完成后，驱动程序往这段内存写10个字节数据，用户进程将这些数据显示出来。**

**驱动程序：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. #include <linux/miscdevice.h>
2. #include <linux/delay.h>
3. #include <linux/kernel.h>
4. #include <linux/module.h>
5. #include <linux/init.h>
6. #include <linux/mm.h>
7. #include <linux/fs.h>
8. #include <linux/types.h>
9. #include <linux/delay.h>
10. #include <linux/moduleparam.h>
11. #include <linux/slab.h>
12. #include <linux/errno.h>
13. #include <linux/ioctl.h>
14. #include <linux/cdev.h>
15. #include <linux/string.h>
16. #include <linux/list.h>
17. #include <linux/pci.h>
18. #include <linux/gpio.h>

21. #define DEVICE\_NAME "mymap"

24. **static** unsigned **char** array[10]={0,1,2,3,4,5,6,7,8,9};
25. **static** unsigned **char** \*buffer;

28. **static** **int** my\_open(**struct** inode \*inode, **struct** file \*file)
29. {
30. **return** 0;
31. }

34. **static** **int** my\_map(**struct** file \*filp, **struct** vm\_area\_struct \*vma)
35. {
36. unsigned **long** page;
37. unsigned **char** i;
38. unsigned **long** start = (unsigned **long**)vma->vm\_start;
39. //unsigned long end =  (unsigned long)vma->vm\_end;
40. unsigned **long** size = (unsigned **long**)(vma->vm\_end - vma->vm\_start);
42. //得到物理地址
43. page = virt\_to\_phys(buffer);
44. //将用户空间的一个vma虚拟内存区映射到以page开始的一段连续物理页面上
45. **if**(remap\_pfn\_range(vma,start,page>>PAGE\_SHIFT,size,PAGE\_SHARED))//第三个参数是页帧号，由物理地址右移PAGE\_SHIFT得到
46. **return** -1;
48. //往该内存写10字节数据
49. **for**(i=0;i<10;i++)
50. buffer[i] = array[i];
52. **return** 0;
53. }

56. **static** **struct** file\_operations dev\_fops = {
57. .owner    = THIS\_MODULE,
58. .open    = my\_open,
59. .mmap   = my\_map,
60. };
62. **static** **struct** miscdevice misc = {
63. .minor = MISC\_DYNAMIC\_MINOR,
64. .name = DEVICE\_NAME,
65. .fops = &dev\_fops,
66. };

69. **static** **int** \_\_init dev\_init(**void**)
70. {
71. **int** ret;
73. //注册混杂设备
74. ret = misc\_register(&misc);
75. //内存分配
76. buffer = (unsigned **char** \*)kmalloc(PAGE\_SIZE,GFP\_KERNEL);
77. //将该段内存设置为保留
78. SetPageReserved(virt\_to\_page(buffer));
80. **return** ret;
81. }

84. **static** **void** \_\_exit dev\_exit(**void**)
85. {
86. //注销设备
87. misc\_deregister(&misc);
88. //清除保留
89. ClearPageReserved(virt\_to\_page(buffer));
90. //释放内存
91. kfree(buffer);
92. }

95. module\_init(dev\_init);
96. module\_exit(dev\_exit);
97. MODULE\_LICENSE("GPL");
98. MODULE\_AUTHOR("LKN@SCUT");

**应用程序：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/dlutbrucezhang/article/details/9080173)

1. #include <unistd.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <fcntl.h>
6. #include <linux/fb.h>
7. #include <sys/mman.h>
8. #include <sys/ioctl.h>
10. #define PAGE\_SIZE 4096

13. **int** main(**int** argc , **char** \*argv[])
14. {
15. **int** fd;
16. **int** i;
17. unsigned **char** \*p\_map;
19. //打开设备
20. fd = open("/dev/mymap",O\_RDWR);
21. **if**(fd < 0)
22. {
23. printf("open fail\n");
24. exit(1);
25. }
27. //内存映射
28. p\_map = (unsigned **char** \*)mmap(0, PAGE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED,fd, 0);
29. **if**(p\_map == MAP\_FAILED)
30. {
31. printf("mmap fail\n");
32. **goto** here;
33. }
35. //打印映射后的内存中的前10个字节内容
36. **for**(i=0;i<10;i++)
37. printf("%d\n",p\_map[i]);

40. here:
41. munmap(p\_map, PAGE\_SIZE);
42. **return** 0;
43. }

**先加载驱动后执行应用程序，用户空间打印如下：**

