张勤一
echo "are you happy ?" | cut -d ' ' -f 3 | tr -d '\r\n '

目录视图

Linux 内存映射函数 mmap（ ）函数详解

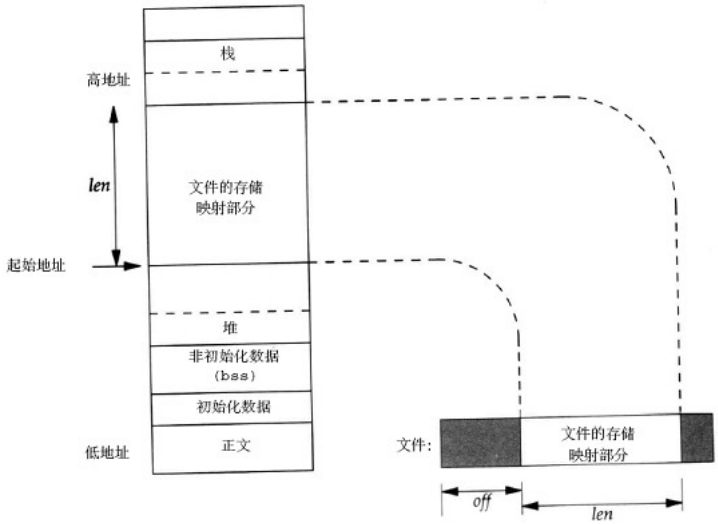
标签： Linux 内存 地图 内核 行业数据

2013-06-12 19:02 17324人阅读 评论

分类： Linux System Programming（17）

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
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
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 Network programming**》卷二
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
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**m**
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
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
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
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**m**
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
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**m**
存映射

[cpp] view plain copy print ?

关闭


阅读：99217

Linux Shell 脚本攻略
文章：32篇
阅读：59521

Linux内核设计的艺术
文章：33篇
阅读：57866

Linux 系统编程
文章：21篇
阅读：67605


Android SDK 样例源码分析
文章：6篇
阅读：45653


android网络编程
文章：7篇
阅读：55398

《UNIX环境高级编程》
文章：56篇
阅读：259299

TCP/IP详解
文章：20篇
阅读：40785


UNIX系统编程
文章：15篇
阅读：35773

Linux C编程
文章：30篇
阅读：167033


Android开发精要
文章：28篇
阅读：144898

文章分类

Android (110)
算法 (45)
Android SDK实例分析 (28)
Android常见运行时错误 (7)
Android布局 (3)
数据库 (2)
SQLite (2)
计算机网络 (19)
Android应用实例分析 (5)
Android四大组件 (1)
Android中的线程处理 (5)
IP (1)
Socket (1)
Android环境搭建 (3)
安卓巴士Demo (6)

```
01. <span style="background-
    color: rgb(255, 255, 255);">void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset
```

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
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
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 plain copy print ?
01. 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 plain copy print ?
01. length:要映射的内存区域的大小
```

```
[cpp] view plain copy print ?
01. 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
02. PROT_EXEC //页内容可以被执行
03. PROT_READ //页内容可以被读取
04. PROT_WRITE //页可以被写入
05. PROT_NONE //页不可访问
```

```
[cpp] view plain copy print ?
01. 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
02. 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
    败。并且起始地址必须落在页的边界上。
03. 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。
04. 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。
05. MAP_DENYWRITE :这个标志被忽略。
06. MAP_EXECUTABLE :同上
07. 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
    违例信号。
08. MAP_LOCKED :锁定映射区的页面，从而防止页面被交换出内存。
09. 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 plain copy print ?
01. fd:文件描述符(由open函数返回)
```

```
[cpp] view plain copy print ?
01. 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_FAILE**
***)-1]**，**munmap**返回**-1**。**errno**被设为以下的某个值

```
[cpp] view plain copy print ?
01. EACCES:访问出错
02. EAGAIN:文件已被锁定，或者太多的内存已被锁定
03. EBADF:fd不是有效的文件描述词
04. EINVAL:一个或者多个参数无效
05. ENFILE:已达到系统对打开文件的限制
06. ENODEV:指定文件所在的文件系统不支持内存映射
07. ENOMEM:内存不足，或者进程已超出最大内存映射数量
08. EPERM:权能不足，操作不允许
09. 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 plain copy print ?
```

关闭

```
01. int munmap(void *start, size_t length)
```

```
[cpp] view plain copy print ?
01. start:要取消映射的内存区域的起始地址
```

```
[cpp] view plain copy print ?
01. 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 plain copy print ?
01. 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(**样你内存的更新就能立即保存到文件里

```
[cpp] view plain copy print ?
01. start:要进行同步的映射的内存区域的起始地址。
02. length:要同步的内存区域的大小
03. flag:flags可以为以下三个值之一:
04. MS_ASYNC : 请Kernel快将资料写入。
05. MS_SYNC : 在msync结束返回前，将资料写入。
06. 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 plain copy print ?
01. #include <linux/miscdevice.h>
02.  #include <linux/delay.h>
03.  #include <linux/kernel.h>
04.  #include <linux/module.h>
05.  #include <linux/init.h>
06.  #include <linux/mm.h>
07.  #include <linux/fs.h>
08.  #include <linux/types.h>
09.  #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>
19.
20.
21.  #define DEVICE_NAME "mymap"
22.
23.
24.  static unsigned char array[10]={0,1,2,3,4,5,6,7,8,9};
25.  static unsigned char *buffer;
26.
27.
28.  static int my_open(struct inode *inode, struct file *file)
29.  {
30.      return 0;
31.  }
32.
33.
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);
41.
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_SHI
46.         return -1;
47.
48.     //往该内存写10字节数据
49.     for(i=0;i<10;i++)
50.         buffer[i] = array[i];
51.
52.     return 0;
53. }
54.
55.
56. static struct file_operations dev_fops = {
57.     .owner    = THIS_MODULE,
58.     .open     = my_open,
59.     .mmap     = my_map,
60. };
61.
62. static struct miscdevice misc = {
63.     .minor = MISC_DYNAMIC_MINOR,
64.     .name = DEVICE_NAME,
65.     .fops = &dev_fops,
66. };
67.
68.
69. static int __init dev_init(void)
70. {
71.     int ret;
72.
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));
79.
80.     return ret;
81. }
82.
83.
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. }
93.
94.
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 plain  copy  print  ?
01. #include <unistd.h>
02.  #include <stdio.h>
03.  #include <stdlib.h>
04.  #include <string.h>
05.  #include <fcntl.h>
06.  #include <linux/fb.h>
07.  #include <sys/mman.h>
08.  #include <sys/ioctl.h>
09.
10.  #define PAGE_SIZE 4096
11.
12.
13.  int main(int argc , char *argv[])
14.  {
15.      int fd;
16.      int i;
```

```
    unsigned char *p_map;

    //打开设备
    fd = open("/dev/mymap",O_RDWR);
    if(fd < 0)
    {
        printf("open fail\n");
        exit(1);
    }

    //内存映射
    p_map = (unsigned char *)mmap(0, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,fd, 0);
    if(p_map == MAP_FAILED)
    {
        printf("mmap fail\n");
        goto here;
    }

    //打印映射后的内存中的前10个字节内容
    for(i=0;i<10;i++)
        printf("%d\n",p_map[i]);


here:
    munmap(p_map, PAGE_SIZE);
    return 0;
}
```

先加载驱动后执行应用程序，用户空间打印如下：

```
0
1
2
3
4
5
6
7
8
9
```

顶 3　　踩 0

- 上一篇　程序（进程）内存分布 解析
- 下一篇　Linux System Programming --Chapter Two

我的同类文章

Linux System Programming（17）

- Linux System Programming -- Appen... 2013-06-18 阅读 1140
- Linux System Programming --Chapte... 2013-06-18 阅读 1277
- 程序（进程）内存分布 解析 2013-06-12 阅读 3139
- Linux System Programming --Chapte... 2013-06-17 阅读 1171
- Linux 开机启动过程分析 2013-06-11 阅读 3466
- 字符编码总结（UTF-8，UNICODE） 2013-06-
- Linux System Programming --Chapte... 2013-06-
- Linux System Programming --Chapte... 2013-06-
- Linux System Programming --Chapte... 2013-06-
- Linux 2.6 中的直接 I/O 技术 2013-06-

更多文章

参考知识库

Linux知识库

11722 关注 | 3939 收录

猜你在找

关闭

从零写Bootloader及移植uboot、…
话说linux内核-uboot和系统移植…
input子系统基础之按键-linux驱…
驱动框架入门之LED-linux驱动开…
字符设备驱动基础-linux驱动开发…
Linux的mmap内存映射机制解析
Linux的mmap文件内存映射机制
Linux内核 内存映射文件机制mmap
linux编程学习笔记三 虚拟内存映…
Linux内存映射文件mmap

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 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

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VP
IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5 Spring Apa
HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Ra
Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App S
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr An
Cloud Foundry Redis Scala Django Bootstrap

关闭