原 Linux共享内存详解（下）

在共享内存（上）中，主要围绕着系统调用mmap()进行讨论的，本部分将讨论系统V共享内存，并通过实验结果对比来阐述两者的异同。系统V共享内存指的是把所有共享数据放在共享内存区域（IPC shared memory region），任何想要访问该数据的进程都必须在本进程的地址空间新增一块内存区域，用来映射存放共享数据的物理内存页面。

系统调用mmap()通过映射一个普通文件实现共享内存。系统V则是通过映射特殊文件系统shm中的文件实现进程间的共享内存通信。也就是说，每个共享内存区域对应特殊文件系统shm中的一个文件（这是通过shmid_kernel结构联系起来的），后面还将阐述。

1、系统V共享内存原理

进程间需要共享的数据被放在一个叫做IPC共享内存区域的地方，所有需要访问该共享区域的进程都要把该共享区域映射到本进程的地址空间中去。系统V共享内存通过shmget获得或创建一个IPC共享内存区域，并返回相应的标识符。内核在保证shmget获得或创建一个共享内存区，初始化该共享内存区相应的 shmid_kernel结构注同时，还将在特殊文件系统shm中，创建并打开一个同名文件，并在内存中建立起该文件的相应dentry及inode结构，新打开的文件不属于任何一个进程（任何进程都可以访问该共享内存区）。所有这一切都是系统调用shmget完成的。

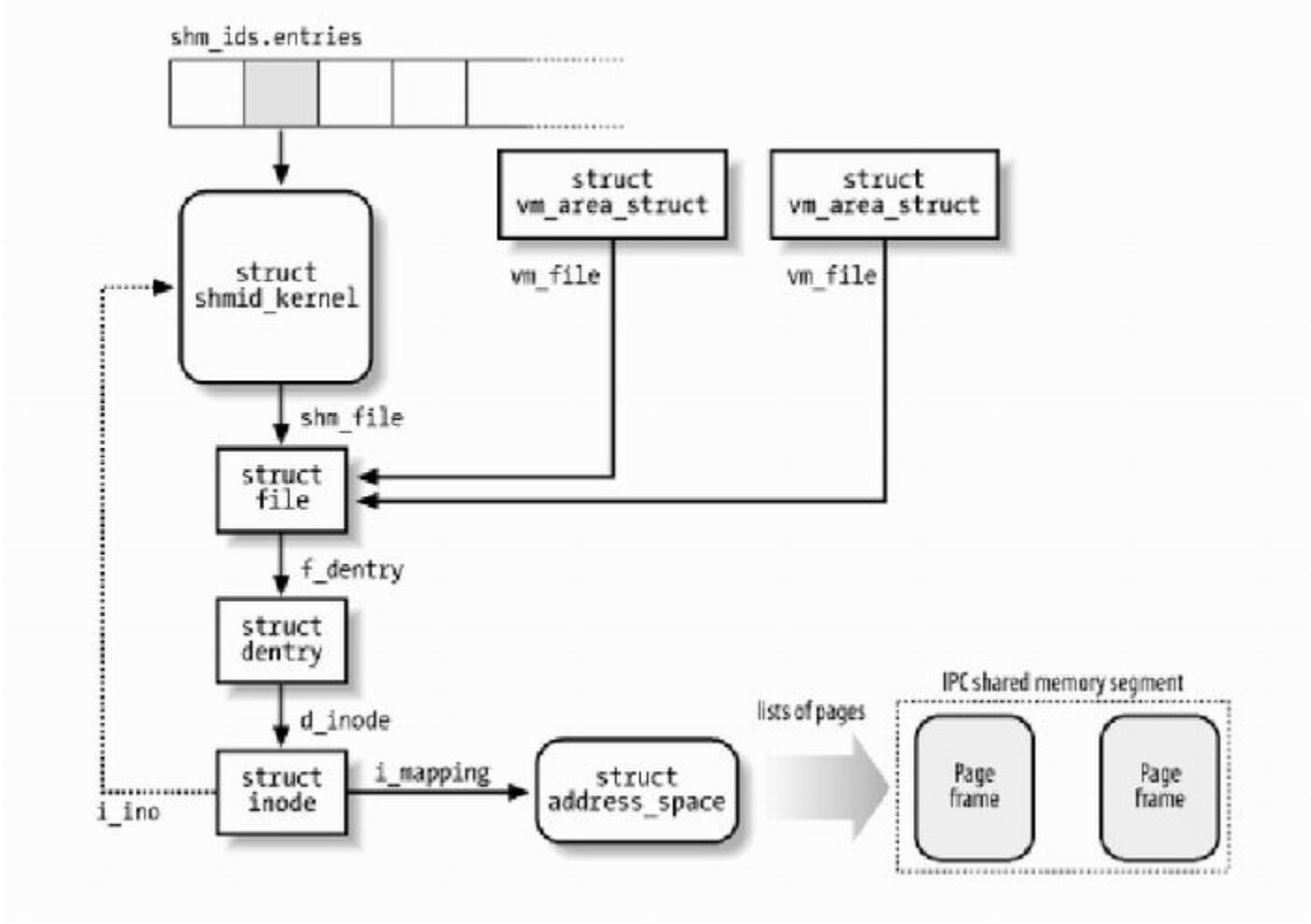
注：每一个共享内存区都有一个控制结构struct shmid_kernel，shmid_kernel是共享内存区域中非常重要的一个数据结构，它是存储管理和文件系统结合起来的桥梁，定义如下：

```
struct shmid_kernel /* private to the kernel */
{
 struct kern_ipc_perm shm_perm;
 struct file *  shm_file;
 int   id;
 unsigned long  shm_nattch;
 unsigned long  shm_segsz;
 time_t   shm_atim;
```

```
 time_t   shm_dtim;
 time_t   shm_ctim;
 pid_t   shm_cprid;
 pid_t   shm_lprid;
};
```

该结构中最重要的一个域应该是shm_file，它存储了将被映射文件的地址。每个共享内存区对象都对应特殊文件系统shm中的一个文件，一般情况下，特殊文件系统shm中的文件是不能用read()、 write()等方法访问的，当采取共享内存的方式把其中的文件映射到进程地址空间后，可直接采用访问内存的方式对其访问。

这里我们采用[1]中的图表给出与系统V共享内存相关数据结构:



正如消息队列和信号灯一样，内核通过数据结构struct ipc_ids shm_ids维护系统中的所有共享内存区域。上图中的shm_ids.entries变量指向一个ipc_id结构数组，而每个ipc_id结构数组中有个指向kern_ipc_perm结构的指针。到这里读者应该很熟悉了，对于系统V共享内存区来说，kern_ipc_perm的宿主是 shmid_kernel结构，shmid_kernel是用来描述一个共享内存区域的，这样内核就能够控制系统中所有的共享区域。同时，在 shmid_kernel结构的file类型指针shm_file指向文件系统shm中相应的文件，这样，共享内存区域就与shm文件系统中的文件对应起来。

在创建了一个共享内存区域后，还要将它映射到进程地址空间，系统调用shmat()完成此项功能。由于在调用 shmget()时，已经创建了文件系统shm中的一个同名文件与共享内存区域相对应，因此，调用shmat()的过程相当于映射文件系统shm中的同名文件过程，原理与mmap()大同小异。

2、系统V共享内存API

对于系统V共享内存，主要有以下几个API：shmget()、shmat()、shmdt()及shmctl()。

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

shmget（）用来获得共享内存区域的ID，如果不存在指定的共享区域就创建相应的区域。shmat()把共享内存区域映射到调用进程的地址空间中去，这样，进程就可以方便地对共享区域进行访问操作。 shmdt()调用用来解除进程对共享内存区域的映射。shmctl实现对共享内存区域的控制操作。这里我们不对这些系统调用作具体的介绍，读者可参考相应的手册页面，后面的范例中将给出它们的调用方法。

注：shmget的内部实现包含了许多重要的系统V共享内存机制； shmat在把共享内存区域映射到进程空间时，并不真正改变进程的页表。当进程第一次访问内存映射区域访问时，会因为没有物理页表的分配而导致一个缺页异常，然后内核再根据相应的存储管理机制为共享内存映射区域分配相应的页表。

3、系统V共享内存限制

在/proc/sys/kernel/目录下，记录着系统V共享内存的一下限制，如一个共享内存区的最大字节数shmmax，系

统范围内最大共享内存区标识符数shmmni等，可以手工对其调整，但不推荐这样做。

在[2]中，给出了这些限制的测试方法，不再赘述。

4、系统V共享内存范例

本部分将给出系统V共享内存API的使用方法，并对比分析系统V共享内存机制与mmap()映射普通文件实现共享内存之间的差异，首先给出两个进程通过系统V共享内存通信的范例：

```
/***** testwrite.c *******/
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
typedef struct{
 char name[4];
 int age;
} people;
main(int argc, char** argv)
{
 int shm_id,i;
 key_t key;
 char temp;
 people *p_map;
 char* name = "/dev/shm/myshm2";
 key = ftok(name,0);
 if(key==-1)
  perror("ftok error");
 shm_id=shmget(key,4096,IPC_CREAT);
 if(shm_id==-1)
 {
  perror("shmget error");
  return;
 }
 p_map=(people*)shmat(shm_id,NULL,0);
 temp='a';
```

```
 for(i = 0;i<10;i++)
 {
  temp+=1;
  memcpy((*(p_map+i)).name,&temp,1);
  (*(p_map+i)).age=20+i;
 }
 if(shmdt(p_map)==-1)
  perror(" detach error ");
}
/********** testread.c ************/
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
typedef struct{
 char name[4];
 int age;
} people;
main(int argc, char** argv)
{
 int shm_id,i;
 key_t key;
 people *p_map;
 char* name = "/dev/shm/myshm2";
 key = ftok(name,0);
 if(key == -1)
  perror("ftok error");
 shm_id = shmget(key,4096,IPC_CREAT);
 if(shm_id == -1)
 {
  perror("shmget error");
  return;
 }
 p_map = (people*)shmat(shm_id,NULL,0);
 for(i = 0;i<10;i++)
 {
 printf( "name:%s\n",(*(p_map+i)).name );
```

```
 printf( "age %d\n",(*(p_map+i)).age );
 }
 if(shmdt(p_map) == -1)
  perror(" detach error ");
}
```

testwrite.c创建一个系统V共享内存区，并在其中写入格式化数据；testread.c访问同一个系统V共享内存区，读出其中的格式化数据。分别把两个程序编译为testwrite及 testread，先后执行./testwrite及./testread 则./testread输出结果如下：

```
name: b age 20; name: c age 21; name: d age 22; name: e age 23; name: f age 24;
name: g age 25; name: h age 26; name: I age 27; name: j age 28; name: k age 29;
```

通过对试验结果分析，对比系统V与mmap()映射普通文件实现共享内存通信，可以得出如下结论：

1、 系统V共享内存中的数据，从来不写入到实际磁盘文件中去；而通过mmap()映射普通文件实现的共享内存通信可以指定何时将数据写入磁盘文件中。注：前面讲到，系统V共享内存机制实际是通过映射特殊文件系统shm中的文件实现的，文件系统shm的安装点在交换分区上，系统重新引导后，所有的内容都丢失。

2、 系统V共享内存是随内核持续的，即使所有访问共享内存的进程都已经正常终止，共享内存区仍然存在（除非显式删除共享内存），在内核重新引导之前，对该共享内存区域的任何改写操作都将一直保留。

3、 通过调用mmap()映射普通文件进行进程间通信时，一定要注意考虑进程何时终止对通信的影响。而通过系统V共享内存实现通信的进程则不然。注：这里没有给出shmctl的使用范例，原理与消息队列大同小异。

结论：

共享内存允许两个或多个进程共享一给定的存储区，因为数据不需要来回复制，所以是最快的一种进程间通信机制。共享内存可以通过mmap()映射普通文件（特殊情况下还可以采用匿名映射）机制实现，也可以通过系统V共享内存机制实现。应用接口和原理很简单，内部机制复杂。为了实现更安全通信，往往还与信号灯等同步机制共同使用。

共享内存涉及到了存储管理以及文件系统等方面的知识，深入理解其内部机制有一定的难度，关键还要紧紧抓住内核使用的重要数据结构。系统V共享内存是以文件的形式组织在特殊文件系统shm中的。通过shmget可以创建或获得共享内存的标识符。取得共享内存标识符后，要通过shmat将这个内存区映射到本进程的虚拟地址空间。

参考资料

[1] Understanding the Linux Kernel, 2nd Edition, By Daniel P. Bovet, Marco Cesati ，对各主题阐述得重点突出，脉络清晰。

[2] UNIX网络编程第二卷：进程间通信，作者：W.Richard Stevens，译者：杨继张，清华大学出版社。对mmap()有详细阐述。

[3] Linux内核源代码情景分析（上），毛德操、胡希明著，浙江大学出版社，给出了mmap()相关的源代码分析。

[4]shmget、shmat、shmctl、shmdt手册

关于作者

郑彦兴，国防科大攻读博士学位。联系方式： mlinux@163.com

发表于 @ 2008年10月22日 10:41:00 | 评论(0) | 举报 | 收藏

旧一篇:Linux共享内存详解（上） | 新一篇:Linux Make工具和Makefile

评论内容:

用 户 名: huapuyu7 匿名评论

发表评论