

## 40 | IPC（上）：不同项目组之间抢资源，如何协调？

2019-06-28 刘超

趣谈Linux操作系统

[进入课程 >](#)



讲述：刘超

时长 10:45 大小 9.86M



我们前面讲了，如果项目组之间需要紧密合作，那就需要共享内存，这样就像把两个项目组放在一个会议室一起沟通，会非常高效。这一节，我们就来详细讲讲这个进程之间共享内存的机制。

有了这个机制，两个进程可以像访问自己内存中的变量一样，访问共享内存的变量。但是同时问题也来了，当两个进程共享内存了，就会存在同时读写的问题，就需要对于共享的内存进行保护，就需要信号量这样的同步协调机制。这些也都是我们这节需要探讨的问题。下面我们就——来看。

共享内存和信号量也是 System V 系列的进程间通信机制，所以很多地方和我们讲过的消息队列有点儿像。为了将共享内存和信号量结合起来使用，我这里定义了一个 share.h 头文件，里面放了一些共享内存和信号量在每个进程都需要的函数。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/ipc.h>
4 #include <sys/shm.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/sem.h>
8 #include <string.h>
9
10 #define MAX_NUM 128
11
12 struct shm_data {
13     int data[MAX_NUM];
14     int datalength;
15 };
16
17 union semun {
18     int val;
19     struct semid_ds *buf;
20     unsigned short int *array;
21     struct seminfo *__buf;
22 };
23
24 int get_shmid(){
25     int shmid;
26     key_t key;
27
28     if((key = ftok("/root/sharememory/sharememorykey", 1024)) < 0){
29         perror("ftok error");
30         return -1;
31     }
32
33     shmid = shmget(key, sizeof(struct shm_data), IPC_CREAT|0777);
34     return shmid;
35 }
36
37 int get_semaphoreid(){
38     int semid;
39     key_t key;
40
41     if((key = ftok("/root/sharememory/semaphorekey", 1024)) < 0){
42         perror("ftok error");
43         return -1;
44     }
45
46     semid = semget(key, 1, IPC_CREAT|0777);
47     return semid;
48 }
49
50 int semaphore_init (int semid) {
```

```

51 union semun argument;
52 unsigned short values[1];
53 values[0] = 1;
54 argument.array = values;
55 return semctl (semid, 0, SETALL, argument);
56 }
57
58 int semaphore_p (int semid) {
59     struct sembuf operations[1];
60     operations[0].sem_num = 0;
61     operations[0].sem_op = -1;
62     operations[0].sem_flg = SEM_UNDO;
63     return semop (semid, operations, 1);
64 }
65
66 int semaphore_v (int semid) {
67     struct sembuf operations[1];
68     operations[0].sem_num = 0;
69     operations[0].sem_op = 1;
70     operations[0].sem_flg = SEM_UNDO;
71     return semop (semid, operations, 1);
72 }


```

## 共享内存

我们先来看里面对于共享内存的操作。

首先，创建之前，我们要有一个 key 来唯一标识这个共享内存。这个 key 可以根据文件系统上的一个文件的 inode 随机生成。

然后，我们需要创建一个共享内存，就像创建一个消息队列差不多，都是使用 `xxxget` 来创建。其中，创建共享内存使用的是下面这个函数：

 复制代码

```

1 int shmget(key_t key, size_t size, int shmflag);


```

其中，key 就是前面生成的那个 key，shmflag 如果为 `IPC_CREAT`，就表示新创建，还可以指定读写权限 `0777`。

对于共享内存，需要指定一个大小 size，这个一般要申请多大呢？一个最佳实践是，我们将多个进程需要共享的数据放在一个 struct 里面，然后这里的 size 就应该是这个 struct 的大小。这样每一个进程得到这块内存后，只要强制将类型转换为这个 struct 类型，就能够访问里面的共享数据了。

在这里，我们定义了一个 struct shm\_data 结构。这里面有两个成员，一个是一个整型的数组，一个是数组中元素的个数。


生成了共享内存以后，接下来就是将这个共享内存映射到进程的虚拟地址空间中。我们使用下面这个函数来进行操作。

 复制代码

```
1 void *shmat(int shm_id, const void *addr, int shmflg);
```

这里面的 shm\_id，就是上面创建的共享内存的 id，addr 就是指定映射在某个地方。如果不指定，则内核会自动选择一个地址，作为返回值返回。得到了返回地址以后，我们需要将指针强制类型转换为 struct shm\_data 结构，就可以使用这个指针设置 data 和 datalength 了。

当共享内存使用完毕，我们可以通过 shmdt 解除它到虚拟内存的映射。

 复制代码

```
1 int shmdt(const void *shmaddr);
```

## 信号量

看完了共享内存，接下来我们再来看信号量。信号量以集合的形式存在的。

首先，创建之前，我们同样需要有一个 key，来唯一标识这个信号量集合。这个 key 同样可以根据文件系统上的一个文件的 inode 随机生成。

然后，我们需要创建一个信号量集合，同样也是使用 xxxget 来创建，其中创建信号量集合使用的是下面这个函数。

```
1 int semget(key_t key, int nsems, int semflg);
```

这里的 key，就是前面生成的那个 key，shmflag 如果为 IPC\_CREAT，就表示新创建，还可以指定读写权限 0777。

这里，nsems 表示这个信号量集合里面有几个信号量，最简单的情况下，我们设置为 1。

信号量往往代表某种资源的数量，如果用信号量做互斥，那往往将信号量设置为 1。这就是上面代码中 semaphore\_init 函数的作用，这里面调用 semctl 函数，将这个信号量集合中的第 0 个信号量，也即唯一的这个信号量设置为 1。

对于信号量，往往要定义两种操作，P 操作和 V 操作。对应上面代码中 semaphore\_p 函数和 semaphore\_v 函数，semaphore\_p 会调用 semop 函数将信号量的值减一，表示申请占用一个资源，当发现当前没有资源的时候，进入等待。semaphore\_v 会调用 semop 函数将信号量的值加一，表示释放一个资源，释放之后，就允许等待中的其他进程占用这个资源。

我们可以用这个信号量，来保护共享内存中的 struct shm\_data，使得同时只有一个进程可以操作这个结构。

你是否记得咱们讲线程同步机制的时候，构建了一个老板分配活的场景。这里我们同样构建一个场景，分为 producer.c 和 consumer.c，其中 producer 也即生产者，负责往 struct shm\_data 塞入数据，而 consumer.c 负责处理 struct shm\_data 中的数据。

下面我们来看 producer.c 的代码。

```
1 #include "share.h"
2
3 int main() {
4     void *shm = NULL;
5     struct shm_data *shared = NULL;
6     int shmid = get_shmid();
7     int semid = get_semaphoreid();
8     int i;
```

```


9
10 shm = shmat(shmid, (void*)0, 0);
11 if(shm == (void*)-1){
12     exit(0);
13 }
14 shared = (struct shm_data*)shm;
15 memset(shared, 0, sizeof(struct shm_data));
16 semaphore_init(semid);
17 while(1){
18     semaphore_p(semid);
19     if(shared->datalength > 0){
20         semaphore_v(semid);
21         sleep(1);
22     } else {
23         printf("how many integers to caculate : ");
24         scanf("%d",&shared->datalength);
25         if(shared->datalength > MAX_NUM){
26             perror("too many integers.");
27             shared->datalength = 0;
28             semaphore_v(semid);
29             exit(1);
30         }
31         for(i=0;i<shared->datalength;i++){
32             printf("Input the %d integer : ", i);
33             scanf("%d",&shared->data[i]);
34         }
35         semaphore_v(semid);
36     }
37 }
38 }

```

在这里面，get\_shmid 创建了共享内存，get\_semaphoreid 创建了信号量集合，然后 shmat 将共享内存映射到了虚拟地址空间的 shm 指针指向的位置，然后通过强制类型转换，shared 的指针指向放在共享内存里面的 struct shm\_data 结构，然后初始化为 0。semaphore\_init 将信号量进行了初始化。

接着，producer 进入了一个无限循环。在这个循环里面，我们先通过 semaphore\_p 申请访问共享内存的权利，如果发现 datalength 大于零，说明共享内存里面的数据没有被处理过，于是 semaphore\_v 释放权利，先睡一会儿，睡醒了再看。如果发现 datalength 等于 0，说明共享内存里面的数据被处理完了，于是开始往里面放数据。让用户输入多少个，然后每个数是什么，都放在 struct shm\_data 结构中，然后 semaphore\_v 释放权利，等待其他的进程将这些数拿去处理。

我们再来看 consumer 的代码。


 复制代码

```
1 #include "share.h"
2
3 int main() {
4     void *shm = NULL;
5     struct shm_data *shared = NULL;
6     int shmid = get_shmid();
7     int semid = get_semaphoreid();
8     int i;
9
10    shm = shmat(shmid, (void*)0, 0);
11    if(shm == (void*)-1){
12        exit(0);
13    }
14    shared = (struct shm_data*)shm;
15    while(1){
16        semaphore_p(semid);
17        if(shared->datalength > 0){
18            int sum = 0;
19            for(i=0;i<shared->datalength-1;i++){
20                printf("%d+",shared->data[i]);
21                sum += shared->data[i];
22            }
23            printf("%d",shared->data[shared->datalength-1]);
24            sum += shared->data[shared->datalength-1];
25            printf("=%d\n",sum);
26            memset(shared, 0, sizeof(struct shm_data));
27            semaphore_v(semid);
28        } else {
29            semaphore_v(semid);
30            printf("no tasks, waiting.\n");
31            sleep(1);
32        }
33    }
34 }
```

在这里面，get\_shmid 获得 producer 创建的共享内存，get\_semaphoreid 获得 producer 创建的信号量集合，然后 shmat 将共享内存映射到了虚拟地址空间的 shm 指针指向的位置，然后通过强制类型转换，shared 的指针指向放在共享内存里面的 struct shm\_data 结构。


接着，consumer 进入了一个无限循环，在这个循环里面，我们先通过 semaphore\_p 申请访问共享内存的权利，如果发现 datalength 等于 0，就说明没什么活干，需要等待。如果发现 datalength 大于 0，就说明有活干，于是将 datalength 个整型数字从 data 数组中取出来求和。最后将 struct shm\_data 清空为 0，表示任务处理完毕，通过 semaphore\_v 释放权利。

通过程序创建的共享内存和信号量集合，我们可以通过命令 ipcs 查看。当然，我们也可以通过 ipcrm 进行删除。

 复制代码

```
1 # ipcs
2 ----- Message Queues -----
3 key          msqid      owner      perms      used-bytes   messages
4 ----- Shared Memory Segments -----
5 key          shmid      owner      perms      bytes       nattch     status
6 0x00016988 32768      root      777       516         0
7 ----- Semaphore Arrays -----
8 key          semid      owner      perms      nsems
9 0x00016989 32768      root      777       1
```

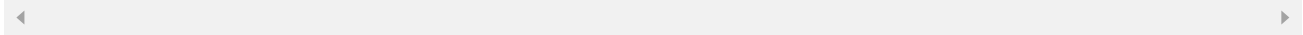
下面我们来运行一下 producer 和 consumer，可以得到下面的结果：

 复制代码

```
1 # ./producer
2 how many integers to caculate : 2
3 Input the 0 integer : 3
4 Input the 1 integer : 4
5 how many integers to caculate : 4
6 Input the 0 integer : 3
7 Input the 1 integer : 4
8 Input the 2 integer : 5
9 Input the 3 integer : 6
10 how many integers to caculate : 7
11 Input the 0 integer : 9
12 Input the 1 integer : 8
13 Input the 2 integer : 7
14 Input the 3 integer : 6
15 Input the 4 integer : 5
16 Input the 5 integer : 4
17 Input the 6 integer : 3
18
19 # ./consumer
```



```
20 3+4=7
21 3+4+5+6=18
22 9+8+7+6+5+4+3=42
```



## 总结时刻

这一节的内容差不多了，我们来总结一下。共享内存和信号量的配合机制，如下图所示：

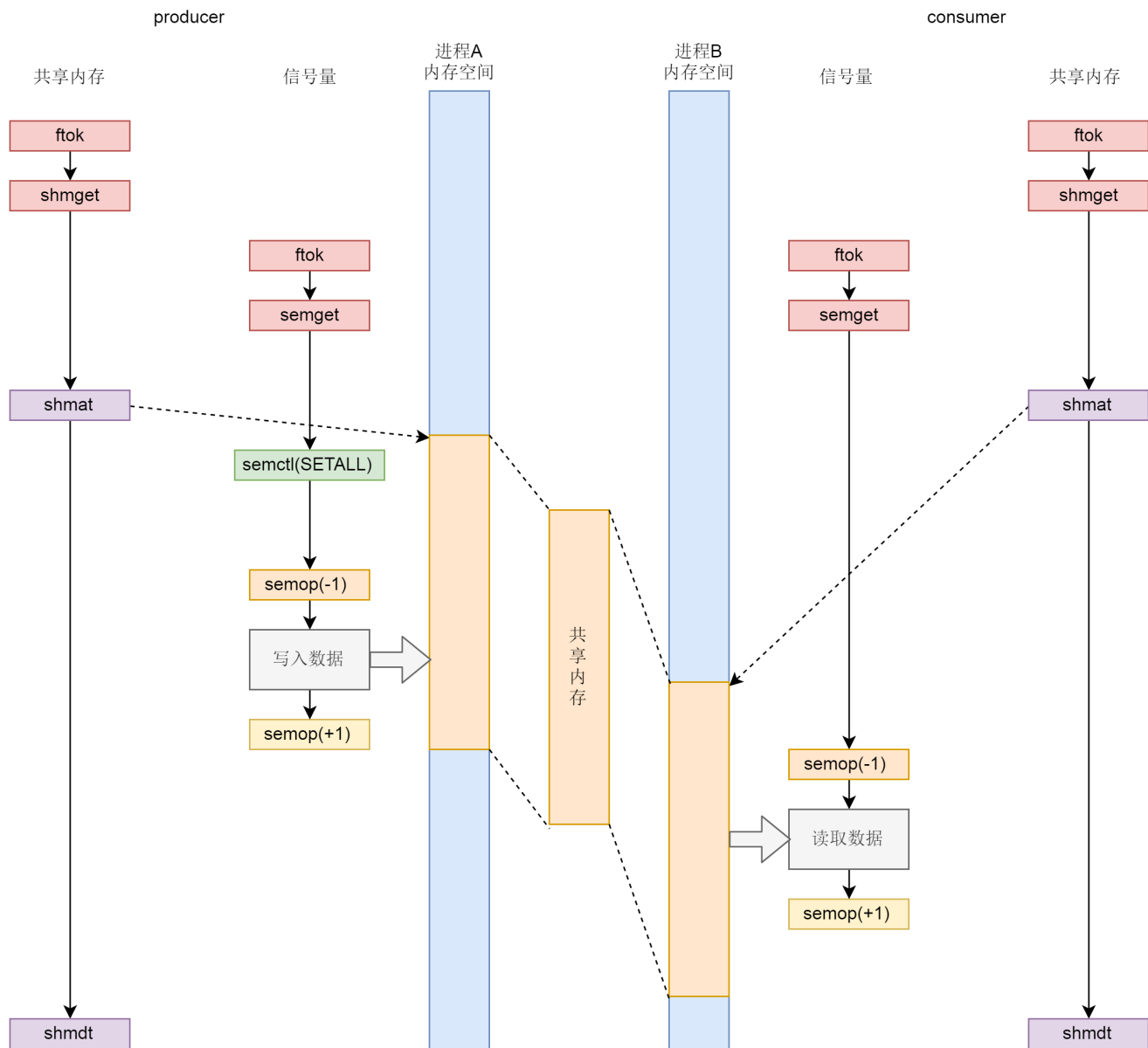
无论是共享内存还是信号量，创建与初始化都遵循同样流程，通过 `ftok` 得到 `key`，通过 `xxxget` 创建对象并生成 `id`；

生产者和消费者都通过 `shmat` 将共享内存映射到各自的内存空间，在不同的进程里面映射的位置不同；

为了访问共享内存，需要信号量进行保护，信号量需要通过 `semctl` 初始化为某个值；

接下来生产者和消费者要通过 `semop(-1)` 来竞争信号量，如果生产者抢到信号量则写入，然后通过 `semop(+1)` 释放信号量，如果消费者抢到信号量则读出，然后通过 `semop(+1)` 释放信号量；

共享内存使用完毕，可以通过 `shmdt` 来解除映射。



## 课堂练习

信号量大于 1 的情况下，应该如何使用？你可以试着构建一个场景。

欢迎留言和我分享你的疑惑和见解，也欢迎可以收藏本节内容，反复研读。你也可以把今天的内容分享给你的朋友，和他一起学习和进步。

# 趣谈 Linux 操作系统

像故事一样的操作系统入门课

刘超

网易杭州研究院

云计算技术部首席架构师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 39 | 管道：项目组A完成了，如何交接给项目组B？

## 精选留言 (5)

写留言



Amark

2019-06-29

如果线程是掉用的到基本单位，那么进程的共享资源呢？



Amark

2019-06-29

请教一个问题，CPU调度是以进程为单位的吗，还是以线程？



莫名

2019-06-28

System V IPC具有很好的移植性，但缺点也比较明显，不能接口自成一套，难以使用现有的fd操作函数。建议对比讲一下比较流行的POSIX IPC。



**许童童**

2019-06-28

信号量大于 1 的情况下，应该如何使用？  
可以让多个进程同时访问一个共享内存。



**Tianz**

2019-06-28

超哥，现在是不是推荐使用 POSIX 系列的 IPC 呢？

