[共享内存函数（shmget、shmat、shmdt、shmctl）及其范例](http://blog.csdn.net/guoping16/article/details/6584058)

标签： [struct](http://www.csdn.net/tag/struct)[null](http://www.csdn.net/tag/null)[cmd](http://www.csdn.net/tag/cmd)[gcc](http://www.csdn.net/tag/gcc)[linux](http://www.csdn.net/tag/linux)[编程](http://www.csdn.net/tag/%e7%bc%96%e7%a8%8b)

2011-07-04 17:38 40371人阅读 [评论](http://blog.csdn.net/guoping16/article/details/6584058#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/guoping16/article/details/6584058#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Linux进程间通信（7） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/guoping16/article/details/6584058)

共享内存函数由shmget、shmat、shmdt、shmctl四个函数组成。下面的表格列出了这四个函数的函数原型及其具体说明。

**1.   shmget函数原型**

|  |  |  |
| --- | --- | --- |
| shmget(得到一个共享内存标识符或创建一个共享内存对象) | | |
| 所需头文件 | #include <sys/ipc.h>  #include <sys/shm.h> | |
| 函数说明 | 得到一个共享内存标识符或创建一个共享内存对象并返回共享内存标识符 | |
| 函数原型 | int shmget(key\_t key, size\_t size, int shmflg) | |
| 函数传入值 | key | 0(IPC\_PRIVATE)：会建立新共享内存对象 |
| 大于0的32位整数：视参数shmflg来确定操作。通常要求此值来源于ftok返回的IPC键值 |
| size | 大于0的整数：新建的共享内存大小，以字节为单位 |
| 0：只获取共享内存时指定为0 |
| shmflg | 0：取共享内存标识符，若不存在则函数会报错 |
| IPC\_CREAT：当shmflg&IPC\_CREAT为真时，如果内核中不存在键值与key相等的共享内存，则新建一个共享内存；如果存在这样的共享内存，返回此共享内存的标识符 |
| IPC\_CREAT|IPC\_EXCL：如果内核中不存在键值与key相等的共享内存，则新建一个消息队列；如果存在这样的共享内存则报错 |
| 函数返回值 | 成功：返回共享内存的标识符 | |
| 出错：-1，错误原因存于error中 | |
| 附加说明 | 上述shmflg参数为模式标志参数，使用时需要与IPC对象存取权限（如0600）进行|运算来确定信号量集的存取权限 | |
| 错误代码 | EINVAL：参数size小于SHMMIN或大于SHMMAX  EEXIST：预建立key所指的共享内存，但已经存在  EIDRM：参数key所指的共享内存已经删除  ENOSPC：超过了系统允许建立的共享内存的最大值(SHMALL)  ENOENT：参数key所指的共享内存不存在，而参数shmflg未设IPC\_CREAT位  EACCES：没有权限  ENOMEM：核心内存不足 | |

在[**Linux**](http://lib.csdn.net/base/linux)环境中，对开始申请的共享内存空间进行了初始化，初始值为0x00。

如果用shmget创建了一个新的消息队列对象时，则shmid\_ds结构成员变量的值设置如下：

        shm\_lpid、shm\_nattach、shm\_atime、shm\_dtime设置为0。

        msg\_ctime设置为当前时间。

        shm\_segsz设成创建共享内存的大小。

        shmflg的读写权限放在shm\_perm.mode中。

        shm\_perm结构的uid和cuid成员被设置成当前进程的有效用户ID，gid和cuid成员被设置成当前进程的有效组ID。

**2.   shmat函数原型**

|  |  |  |
| --- | --- | --- |
| shmat(把共享内存区对象映射到调用进程的地址空间) | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/shm.h> | |
| 函数说明 | 连接共享内存标识符为shmid的共享内存，连接成功后把共享内存区对象映射到调用进程的地址空间，随后可像本地空间一样访问 | |
| 函数原型 | void \*shmat(int shmid, const void \*shmaddr, int shmflg) | |
| 函数传入值 | msqid | 共享内存标识符 |
| shmaddr | 指定共享内存出现在进程内存地址的什么位置，直接指定为NULL让内核自己决定一个合适的地址位置 |
| shmflg | SHM\_RDONLY：为只读模式，其他为读写模式 |
| 函数返回值 | 成功：附加好的共享内存地址 | |
| 出错：-1，错误原因存于error中 | |
| 附加说明 | fork后子进程继承已连接的共享内存地址。exec后该子进程与已连接的共享内存地址自动脱离(detach)。进程结束后，已连接的共享内存地址会自动脱离(detach) | |
| 错误代码 | EACCES：无权限以指定方式连接共享内存  EINVAL：无效的参数shmid或shmaddr  ENOMEM：核心内存不足 | |

**3.   shmdt函数原型**

|  |  |
| --- | --- |
| shmat(断开共享内存连接) | |
| 所需头文件 | #include <sys/types.h>  #include <sys/shm.h> |
| 函数说明 | 与shmat函数相反，是用来断开与共享内存附加点的地址，禁止本进程访问此片共享内存 |
| 函数原型 | int shmdt(const void \*shmaddr) |
| 函数传入值 | shmaddr：连接的共享内存的起始地址 |
| 函数返回值 | 成功：0 |
| 出错：-1，错误原因存于error中 |
| 附加说明 | 本函数调用并不删除所指定的共享内存区，而只是将先前用shmat函数连接（attach）好的共享内存脱离（detach）目前的进程 |
| 错误代码 | EINVAL：无效的参数shmaddr |

**4.   shmctl函数原型**

|  |  |  |
| --- | --- | --- |
| shmctl(共享内存管理) | | |
| 所需头文件 | #include <sys/types.h>  #include <sys/shm.h> | |
| 函数说明 | 完成对共享内存的控制 | |
| 函数原型 | int shmctl(int shmid, int cmd, struct shmid\_ds \*buf) | |
| 函数传入值 | msqid | 共享内存标识符 |
| cmd | IPC\_STAT：得到共享内存的状态，把共享内存的shmid\_ds结构复制到buf中 |
| IPC\_SET：改变共享内存的状态，把buf所指的shmid\_ds结构中的uid、gid、mode复制到共享内存的shmid\_ds结构内 |
| IPC\_RMID：删除这片共享内存 |
| buf | 共享内存管理结构体。具体说明参见共享内存内核结构定义部分 |
| 函数返回值 | 成功：0 | |
| 出错：-1，错误原因存于error中 | |
| 错误代码 | EACCESS：参数cmd为IPC\_STAT，确无权限读取该共享内存  EFAULT：参数buf指向无效的内存地址  EIDRM：标识符为msqid的共享内存已被删除  EINVAL：无效的参数cmd或shmid  EPERM：参数cmd为IPC\_SET或IPC\_RMID，却无足够的权限执行 | |

共享内存应用范例

**5.   父子进程通信范例**

父子进程通信范例，shm.c源代码如下：

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <error.h>

#define SIZE 1024

int main()

{

    int shmid ;

    char \*shmaddr ;

    struct shmid\_ds buf ;

    int flag = 0 ;

    int pid ;

    shmid = shmget(IPC\_PRIVATE, SIZE, IPC\_CREAT|0600 ) ;

    if ( shmid < 0 )

    {

            perror("get shm  ipc\_id error") ;

            return -1 ;

    }

    pid = fork() ;

    if ( pid == 0 )

    {

        shmaddr = (char \*)shmat( shmid, NULL, 0 ) ;

        if ( (int)shmaddr == -1 )

        {

            perror("shmat addr error") ;

            return -1 ;

        }

        strcpy( shmaddr, "Hi, I am child process!\n") ;

        shmdt( shmaddr ) ;

        return  0;

    } else if ( pid > 0) {

        sleep(3 ) ;

        flag = shmctl( shmid, IPC\_STAT, &buf) ;

        if ( flag == -1 )

        {

            perror("shmctl shm error") ;

            return -1 ;

        }

        printf("shm\_segsz =%d bytes\n", buf.shm\_segsz ) ;

        printf("parent pid=%d, shm\_cpid = %d \n", getpid(), buf.shm\_cpid ) ;

        printf("chlid pid=%d, shm\_lpid = %d \n",pid , buf.shm\_lpid ) ;

        shmaddr = (char \*) shmat(shmid, NULL, 0 ) ;

        if ( (int)shmaddr == -1 )

        {

            perror("shmat addr error") ;

            return -1 ;

        }

        printf("%s", shmaddr) ;

        shmdt( shmaddr ) ;

        shmctl(shmid, IPC\_RMID, NULL) ;

    }else{

        perror("fork error") ;

        shmctl(shmid, IPC\_RMID, NULL) ;

    }

    return 0 ;

}

编译 gcc shm.c –o shm。

执行 ./shm，执行结果如下：

shm\_segsz =1024 bytes

shm\_cpid = 9503

shm\_lpid = 9504

Hi, I am child process!

**6.   多进程读写范例**

多进程读写即一个进程写共享内存，一个或多个进程读共享内存。下面的例子实现的是一个进程写共享内存，一个进程读共享内存。

（1）下面程序实现了创建共享内存，并写入消息。

shmwrite.c源代码如下：

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#include <string.h>

typedef struct{

    char name[8];

    int age;

} people;

int main(int argc, char\*\* argv)

{

    int shm\_id,i;

    key\_t key;

    char temp[8];

    people \*p\_map;

    char pathname[30] ;

    strcpy(pathname,"/tmp") ;

    key = ftok(pathname,0x03);

    if(key==-1)

    {

        perror("ftok error");

        return -1;

    }

    printf("key=%d\n",key) ;

    shm\_id=shmget(key,4096,IPC\_CREAT|IPC\_EXCL|0600);

    if(shm\_id==-1)

    {

        perror("shmget error");

        return -1;

    }

    printf("shm\_id=%d\n", shm\_id) ;

    p\_map=(people\*)shmat(shm\_id,NULL,0);

    memset(temp, 0x00, sizeof(temp)) ;

    strcpy(temp,"test") ;

    temp[4]='0';

    for(i = 0;i<3;i++)

    {

        temp[4]+=1;

        strncpy((p\_map+i)->name,temp,5);

        (p\_map+i)->age=0+i;

    }

    shmdt(p\_map) ;

    return 0 ;

}

（2）下面程序实现从共享内存读消息。

shmread.c源代码如下：

#include <stdio.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

typedef struct{

    char name[8];

    int age;

} people;

int main(int argc, char\*\* argv)

{

    int shm\_id,i;

    key\_t key;

    people \*p\_map;

    char pathname[30] ;

    strcpy(pathname,"/tmp") ;

    key = ftok(pathname,0x03);

    if(key == -1)

    {

        perror("ftok error");

        return -1;

    }

    printf("key=%d\n", key) ;

    shm\_id = shmget(key,0, 0);

    if(shm\_id == -1)

    {

        perror("shmget error");

        return -1;

    }

    printf("shm\_id=%d\n", shm\_id) ;

    p\_map = (people\*)shmat(shm\_id,NULL,0);

    for(i = 0;i<3;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");

        return -1;

    }

    return 0 ;

}

（3）编译与执行

①    编译gcc shmwrite.c -o  shmwrite。

②    执行./shmwrite，执行结果如下：

key=50453281

shm\_id=688137

③    编译gcc shmread.c -o shmread。

④    执行./shmread，执行结果如下：

key=50453281

shm\_id=688137

name:test1

age 0

name:test2

age 1

name:test3

age 2

⑤    再执行./shmwrite，执行结果如下：

key=50453281

shmget error: File exists

⑥    使用ipcrm -m 688137删除此共享内存。

      摘录自《深入浅出Linux工具与编程》