Linux IPC 之共享内存

System V 共享内存机制： shmget shmat shmdt shmctl

原理及实现：system V IPC 机制下的共享内存本质是一段特殊的内存区域，进程间需要共享的数据被放在该共享内存区域中，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。这样一个使用共享内存的进程可以将信息写入该空间，而另一个使用共享内存的进程又可以通过简单的内存读操作获取刚才写入的信息，使得两个不同进程之间进行了一次信息交换，从而实现进程间的通信。共享内存允许一个或多个进程通过同时出现在它们的虚拟地址空间的内存进行通信，而这块虚拟内存的页面被每个共享进程的页表条目所引用，同时并不需要在所有进程的虚拟内存都有相同的地址。进程对象对于共享内存的访问通过 key（键）来控制，同时通过 key 进行访问权限的检查。

函数定义如下：

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
key_t ftok(const char *pathname, int proj_id);
int shmget(key_t key, int size, int shmflg);
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

函数 ftok 用于创建一个关键字，可以用该关键字关联一个共享内存段。

参数 pathname 为一个全路径文件名，并且该文件必须可访问。

参数 proj_id 通常传入一非 0 字符

通过 pathname 和 proj_id 组合可以创建唯一的 key

如果调用成功，返回一关键字，否则返回-1

函数 shmget 用于创建或打开一共享内存段，该内存段由函数的第一个参数唯一创建。函数成功，则返回一个唯一的共享内存标识号（相当于进程号，唯一的标识着共享内存），失败返回-1。

参数 key 是一个与共享内存段相关联关键字，如果事先已经存在一个与指定关键字关联的共享内存段，则直接返回该内存段的标识，表示打开，如果不存在，则创建一个新的共享内存段。key 的值既可以用 ftok 函数产生，也可以是 IPC_PRIVATE（用于创建一个只属于创建进程的共享内存，主要用于父子通信）,表示总是创建新的共享内存段；

参数 size 指定共享内存段的大小，以字节为单位；

参数 shmflg 是一掩码合成值，可以是访问权限值与(IPC_CREAT 或 IPC_EXCL)的合成。IPC_CREAT 表示如果不存在该内存段，则创建它。IPC_EXCL 表示如果该内存段存在，则函数返回失败结果(-1)。如果调用成功，返回内存段标识，否则返回-1

函数 shmat 将共享内存段映射到进程空间的某一地址。

参数 shmid 是共享内存段的标识 通常应该是 shmget 的成功返回值

参数 shmaddr 指定的是共享内存连接到当前进程中的地址位置。通常是 NULL，表

示让系统来选择共享内存出现的地址。

参数 shmflg 是一组位标识，通常为 0 即可。

如果调用成功，返回映射后的进程空间的首地址，否则返回(char *)-1。

函数 shmdt 用于将共享内存段与进程空间分离。

参数 shmaddr 通常为 shmat 的成功返回值。

函数成功返回 0，失败时返回-1.注意，将共享内存分离并没删除它，只是使得该共享内存对当前进程不在可用。

函数 shmctl 是共享内存的控制函数，可以用来删除共享内存段。

参数 shmid 是共享内存段标识 通常应该是 shmget 的成功返回值

参数 cmd 是对共享内存段的操作方式，可选为 IPC_STAT,IPC_SET,IPC_RMID。通常为 IPC_RMID，表示删除共享内存段。

参数 buf 是表示共享内存段的信息结构体数据，通常为 NULL。

有进程连接，执行返回 0，标记删除成功，但是最后一个进程解除连接后，共享内存真正被删除。

例如 shmctl(kshareMem,IPC_RMID)表示删除调共享内存段 kHareMem

示例：有亲缘关系

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define    PERM    S_IRUSR | S_IWUSR    //表示用户可读可写  即 0600

int main(int argc,char **argv)
{
    /*在两个有亲属关系进程间通信,KEY 采用 IPC_PRIVATE 由系统自选*/
    int shmid = shmget(IPC_PRIVATE, 1024, PERM); //只有 IPC_PRIVATE 情况可以不设置 IPC_CREAT
    if(shmid == -1) /*  创建 byte  的共享内存*/
    {
        fprintf(stderr,"Create Share Memory Error:%s\n\a",strerror(errno));
        exit(1);
    }
    if(fork() > 0)    /*  父进程代码*/
    {
        char *p_addr = (char*)shmat(shmid, NULL, 0);    //获得该段共享内存的首地址
        memset(p_addr, '\0', 1024);    //初始化为 0
        strncpy(p_addr, "share memory", 1024);    //存入（写入）内容
        printf("parent %d    Write buffer: %s\n", getpid(), p_addr);
```

```
            sleep(2);
            wait(NULL);  //防止僵尸进程
            shmctl(shmid, IPC_RMID, 0);/*删除共享内存,用 ipcs -m 看共享内存*/
            exit(0);
        }

        else /*  子进程代码*/
        {
            sleep(5);     //让父有足够的时间写
            char *c_addr = (char*)shmat(shmid, NULL, 0);   //取出（读出）内容
            printf("Client pid=%d,shmid =%d Read buffer: %s\n",getpid(),shmid,c_addr);
            exit(0);
        }
}
```

Example:非亲进程间通信的实现步骤如下：

写内存端

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>   //头文件包含
#include <sys/types.h>
main()
{
    key_t key = ftok("b.dat",1);                    //1.  写入端先用 ftok 函数获得 key
    if(key == -1)
    {
        perror("ftok");
        exit(-1);
    }
    int shmid = shmget(key,4096,IPC_CREAT);       //2.  写入端用 shmget 函数创建一共
享内存段
    if(shmid == -1)
    {
        perror("shmget");
        exit(-1);
    }
    char *pMap = (char *)shmat(shmid, NULL, 0);    //3.  获得共享内存段的首地址
    memset(pMap, 0, 4096);
    memcpy(pMap, "hello world", 4096);             //4.  往共享内存段中写入内容
    if(shmdt(pMap) == -1)                          //5.  关闭共享内存段
    {
        perror("shmdt");
```

```
            exit(-1);
        }
    }
```

读内存端：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
main()
{
    key_t key = ftok("b.dat",1);                    //1. 读入端用 ftok 函数获得 key
    if(key == -1)
    {
        perror("ftok");
        exit(-1);
    }
    int shmid = shmget(key,4096,0600|IPC_CREAT);          //2. 读入端用 shmget 函数
打开共享内存段
    if(shmid == -1)
    {
        perror("shmget");
        exit(-1);
    }
    char *pMap = (char *)shmat(shmid, NULL, 0);   //3. 获得共享内存段的首地址
    printf("receive the data:%s\n",pMap);         //4. 读取共享内存段中的内容
    if(shmctl(shmid, IPC_RMID, 0) == -1)          //5. 删除共享内存段
    {
        perror("shmctl");
        exit(-1);
    }
}
```

示例，通过共享内存实现两个程序间的对话。

程序 1，shmwr.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ipc.h>
```

```
#include <sys/shm.h>
struct text
{
        int useful;
        char buf[1024];
};
int main()
{
        int shmid = shmget((key_t)5080 , sizeof(struct text),0600|IPC_CREAT);
        printf("%d \n" ,shmid);
        struct text* ptext = (struct text *)shmat(shmid , NULL , 0);
//      ptext->useful = 0 ;
        while(1)
        {
                if(ptext -> useful == 0)
                {
                        int iret = read(STDIN_FILENO , ptext->buf , 1024);
                        ptext->useful = 1;
                        if(strncmp("end" , ptext->buf,3)==0)
                        {
                                break ;
                        }
                        //ptext ->useful = 0 ;
                }
                sleep(1);
        }
                shmdt((void *)ptext);
                return 0 ;
}
```

程序 2 shmrd.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
struct text
{
        int useful;
        char buf[1024];
};
```

```
int main()
{
        int shmid = shmget((key_t)5080 , sizeof(struct text),0600|IPC_CREAT);
        struct text* ptext = (struct text *)shmat(shmid , NULL , 0);
        ptext->useful = 0 ;
        while(1)
        {
                if(ptext -> useful == 1)
                {
                        write(STDOUT_FILENO,ptext -> buf , strlen(ptext -> buf));
                        ptext ->useful = 0 ;
                        if(strncmp("end" , ptext->buf,3)==0)
                        {
                                break ;
                        }
                }
                sleep(1);
        }
                shmdt((void *)ptext);
                shmctl(shmid , IPC_RMID,0);
                return 0 ;
}
```

shmid_ds 结构体

```
 struct shmid_ds {
                struct ipc_perm shm_perm;       /* Ownership and permissions */
                size_t          shm_segsz;      /* Size of segment (bytes) */
                time_t          shm_atime;      /* Last attach time */
                time_t          shm_dtime;      /* Last detach time */
                time_t          shm_ctime;      /* Last change time */
                pid_t           shm_cpid;       /* PID of creator */
                pid_t           shm_lpid;       /* PID of last shmat(2)/shmdt(2) */
                shmatt_t        shm_nattch;     /* No. of current attaches */
                ...
            };
struct ipc_perm {
                key_t          __key;       /* Key supplied to shmget(2) */
                uid_t          uid;         /* Effective UID of owner */
                gid_t          gid;         /* Effective GID of owner */
                uid_t          cuid;        /* Effective UID of creator */
                gid_t          cgid;        /* Effective GID of creator */
                unsigned short mode;        /* Permissions + SHM_DEST and
                                               SHM_LOCKED flags */
                unsigned short __seq;       /* Sequence number */
```

};

ipc_perm 的 mode 详解表

操作者	读	写（更改 更新）	操作者	读	写（更改 更新）
用户	0400	0200	其他	0004	0002
组	0040	0020			

查看系统共享内存
ipcs

删除共享内存
ipcrm -m shmid