共享内存的具体实现 同一块物理内存被映射到进程A、B各自的进程地址空间。进程A可以即时看到进程B对共享内存中数据的更新，反之亦然。由于多个进程共享同一块内存区域，必然需要某种同步机制，互斥锁和信号量都可以。

采用共享内存通信的一个显而易见的好处是效率高，因为进程可以直接读写内存，而不需要任何数据的拷贝。对于像管道和消息队列等通信方式，则需要在内核和用户空间进行四次的数据拷贝，而共享内存则只拷贝两次数据：一次从输入文件到共享内存区，另一次从共享内存区到输出文件。实际上，进程之间在共享内存时，并不总是读写少量数据后就解除映射，有新的通信时，再重新建立共享内存区域。而是保持共享区域，直到通信完毕为止，这样，数据内容一直保存在共享内存中，并没有写回文件。共享内存中的内容往往是在解除映射时才写回文件的。因此，采用共享内存的通信方式效率是非常高的。

共享内存实现的步骤：

1.创建共享内存，这里用到的函数是shmget，也就是从内存中获得一段共享内存区域。

2.映射共享内存，也就是把这段创建的共享内存映射到具体的进程空间中去，这里使用的函数是shmat。

3.使用不带缓冲的I/O读写命令对其进行操作。

4.撤销映射的操作，其函数为shmdt。

特别提醒：共享内存并未提供同步机制，也就是说，在第一个进程结束对共享内存的写操作之前，并无自动机制可以阻止第二个进程开始对它进行读取。所以我们通常需要用其他的机制来同步对共享内存的访问，例如前面说到的信号量。有关信号量的更多内容，可以查阅我的另一篇文章：Linux进程间通信——使用信号量

5.2 共享内存函数

函数原型：

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

shmget函数：用于开辟或指向一块共享内存，返回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。

keyt key： 共享内存的标识符。如果是父子关系的进程间通信的话，这个标识符用IPC_PRIVATE来代替。

如果两个进程没有任何关系，所以就用ftok()算出来一个标识符（或者自己定义一个）使用了。

int size： 以字节为单位指定需要共享的内存容量。

int flag： 包含9个比特的权限标志，它是这块内存的模式(mode)以及权限标识。

模式可取如下值：

IPC_CREAT 新建（如果已创建则返回目前共享内存的id）

IPC_EXCL 与 IPC_CREAT结合使用，如果已创建则返回错误

将“模式” 和“权限标识”进行或运算，做为第三个参数。如：IPC_CREAT | IPC_EXCL | 0640

其中0640为权限标识,4/2/1 分别表示读/写/执行3种权限，第一个0是UID,第一个6（4+2）表示拥

有者的权限，第二个4表示同组权限，第3个0表示他人的权限。

函数调用成功时返回共享内存的ID，失败时返回-1。

注：创建共享内存时，shmflg参数至少需要 IPC_CREAT | 权限标识，如果只有IPC_CREAT 则申请的地址都是

k=0xffffffff，不能使用；

shmat函数：用来允许本进程访问一块共享内存的函数。

第一次创建共享内存时，它不能任何进程访问，要想启用对该共享内存的访问，必须将其连接到一个进程的地址空间中。

shmat函数就是用来完成此工作的。

int shmid ： 共享内存的ID，即共享内存的标识。

char *shmaddr： 共享内存连接到进程中的起始地址，如果shmaddr为NULL，内核会把共享内存映射到系统选定的地

址空间中；如果shmaddr不为NULL，内核会把共享内存映射到shmaddr指定的位置。

注：一般情况下我们很少需要控制共享内存连接的地址，通常都是让系统来选择一个地址，否则就会使应

用程序对硬件的依赖性过高。所以一般把shmaddr设为NULL。

int shmflag ： 本进程对该内存的操作模式，可以由两个取值：SHM_RND和SHM_RDONLY。SHM_RND为读写模式，

SHM_RDONLY是只读模式。需要注意的是，共享内存的读写权限由它的属主、它的访问权限和当

前进程的属主共同决定。如果当shmflg & SM_RDONLY为true时，即使该共享内存的访问权限允许写操

作，它也不能被写入。该参数通常会被设为0。

函数调用成功时，返回共享内存的起始地址，失败时返回-1。

shmdt函数：用于函数删除本进程对这块内存的使用。

shmdt()与shmat()相反，是用来禁止本进程访问一块共享内存的函数。

char *shmaddr 是那块共享内存的起始地址。

函数调用成功时返回0，失败时返回-1。

shmctl函数： 控制对这块共享内存的使用。

int shmid： 共享内存的ID，即共享内存标识。

int cmd ： 控制命令，表示要采取的动作，可取值如下：

IPC_STAT 得到共享内存的状态：把shmid_ds结构中的数据设置为共享内存的当前关联值

IPC_SET　改变共享内存的状态：把共享内存的当前关联值设置为shmid_ds结构中给出的值

IPC_RMID　删除共享内存段

shmid_ds结构至少包含以下成员：

```
struct shmid_ds {
    uid_t shm_perm.uid;
    uid_t shm_perm.gid;
    uid_t shm_perm.mode;
}
```

struct shmid_ds *buf：　一个结构体指针。IPC_STAT的时候，取得的状态放在这个结构体中。

如果要改变共享内存的状态，用这个结构体指定。

函数调用成功时返回0，失败时返回-1。

说明：别的 ipc 通信或多或少需要在内存中进行多次拷贝，而共享内存是不用的，所以他效率最高