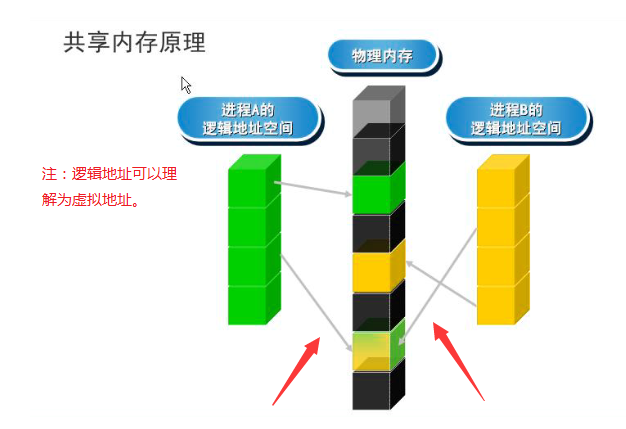
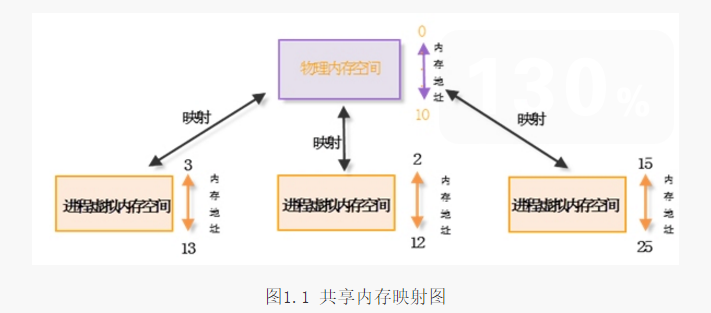
**进程之间的通讯之共享内存**

1. **简介**

**刚刚我们了解了我们的IPC对象，我们知道我们的System V进程间的通信，在系统建立IPC通信的时候，必须指定一个ID值。而该ID的值，我们就可以通过ftok()函数来间接的得到。共享内存就是我们的进程间的一种通信方式。**

**顾名思义，共享内存就是就是允许两个不相关的进程访问同一个物理内存。可以理解为多个进程共享同一块物理内存。共享内存是进程间共享数据的一种最快的方法，进程可以将相同的物理内存，映射到不同的虚拟地址空间中。所有的进程都可以访问共享内存中的数据。可以理解为C语言的malloc分配了一个空间，定义两个指针变量保存了堆区的空间一样。如果一个进行向共享内存中写入了数据，那么它的举动会影响到可以访问同一段内存的其他进程。如下图所示。**





1. **共享内存的实现步骤**

**我们的共享内存的操作步骤分为以下四步:**

**<1>****创建共享内存**

**<2>映射共享内存，即把指定的共享内存映射到多个进程的地址空间，方便进程的访问。**

**<3>撤销共享内存的映射**

**<4>删除共享内存**

1. **创建共享内存 ，既然叫做共享内存，顾名思义，肯定是share memory 的缩写。函数如下。**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**int shmget(key\_t key, size\_t size, int shmflg);**

**功能:申请一块指定大小共享内存**

**参数:**

**@ key IPC\_PRIVATE : 用于亲缘间进程的通信**

**ftok()函数获得: 用于非亲缘关系的进程**

**@size 申请共享内存的大小**

**(注：所有的内存分配时以也4K的倍数为大小进行分配的。即如果一个进程申 请了1块只有1byte的内存，操作系统也会给该内存分配4096bytes。但是真正能够使用的只有1byte。 )**

**@shmflg 权限标志 (常用如下)。**

**IPC\_CREAT | 0666 如果共享内存不存在，则创建一个共享内存，**

**否则直接打开已存在的，返回其ID。**

**IPC\_CREAT | IPC\_EXCL |0666      只有在共享内存不存在的时候，新的共享内存才建立，否则若是存在，shmges调用失败，**

**并设置EEXITST错误码。**

**返回值:**

**成功返回共享内存的id号，失败返回-1**

**-------------------------------------------------------------------------------**

**查看IPC对象**

**ipcs –m 显示共享内存段的信息**

**ipcs -q 显示消息队列段的信息**

**ipcs -s 显示信号灯集段的信息。**

**删除IPC对象**

**ipcrm -m/-q/-s ID**

**理解方法：可以想象成我们使用open函数打开一个文件的时候，若是使用**

**O\_CREAT | O\_EXECL,0666，若是文件存在，则显示打开失败。**

**练习：**

**自己利用ftok（）函数创建一个key值，然后利用shmget()创建共享内存。**

**如果共享内存存在则报错，不存在则创建。自己利用ipcs命令查看共享内存信息。**

1. **映射共享内存，把共享内存和进程的地址空间联系起来。（share memeor attach）**

**手册阅读：**

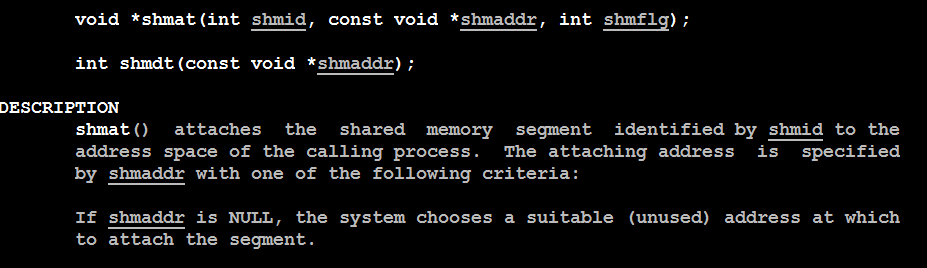
**翻译：**

**<1>函数功能及参数**

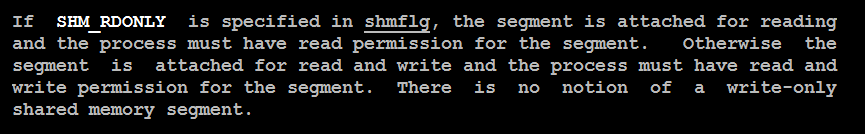
**shmat() 函数映射一个共享内存段，把它和由当前进程调用的由shmid参数指定的地址空间联系起来。**

**这个指定的地址空间，由shmaddr下列选择指定：**

**如果 shmaddr 是NULL，操作系统选择一个合适的（未使用的）共享内存段。**

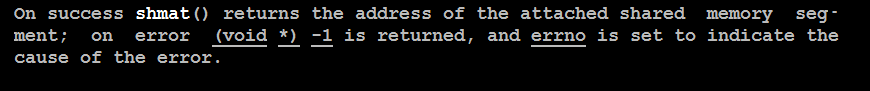


**如果SHM\_RDONLY 被shmflag标志指定，这个被映射的进程的地址空间必须拥有读权限。另一方面，段连接的地址空间若是想要读写的话，必须要有读和写的权限。没有一个只写概念的共享内存段。**



**<2>返回值**

**Shamt()函数成功返回共享内存映射的地址空间，失败返回(void \*)-1并设置error**



**<3>映射共享内存，把共享内存和进程的地址空间联系起来。**

**void \* shmat(int shmid, const void \*shmaddr, int shmflg);**

**功能:把shmid创建共享内存块附加到进程的私有地址。[进程的虚拟地址空间]**

**参数:**

**@ shmid 共享内存段的标识 [由shmget()函数得到]**

**@ shmaddr[将共享内存映射到指定的地址空间]**

**NULL 让系统自动完成映射**

**@ shmflg[映射的标志] 0 映射可以读写;**

**SHM\_RDONLY 映射后只能读**

**返回值：成功返回映射后成功返回映射后的共享内存的地址**

**失败返回（void \*）-1,并且置errno**

**注意：进程结束之后，共享内存的映射自动撤销。**

1. **撤销共享内存。**

**int shmdt (const void \* shmaddr);**

**功能：撤销共享内存到进程地址空间的映射**

**参数：**

**@smaddr 共享内存映射到进程指定的地址空间**

**返回值:**

**成功返回 0**

**失败返回 -1， 并且置errno**

**注:给shmdt传递的地址必须是shmat()函数获得的。**

1. **删除共享内存。[shmat control]**

**int shmctl(int shmid, int cmd,struct shmid\_ds \*buf);**

**功能：对共享内存进行控制**

**参数：**

**@shmid 共享内存段的标识 [由shmget()函数得到]**

**@cmd 共享内存的控制命令**

**IPC\_RMID 删除共享内存。**

**@buf shmid 的一些信息。**

**NULL 表示不需要使用它。**

**返回值：**

**成功返回0，失败返回-1**

**//结构体简介 [了解即可]**

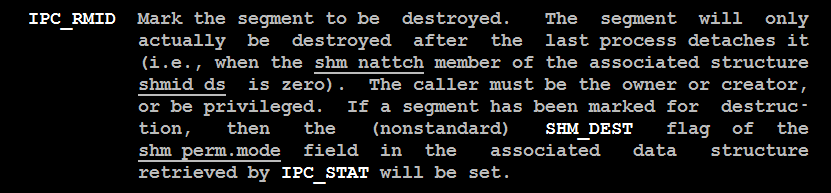
**struct shmid\_ds**

**{**

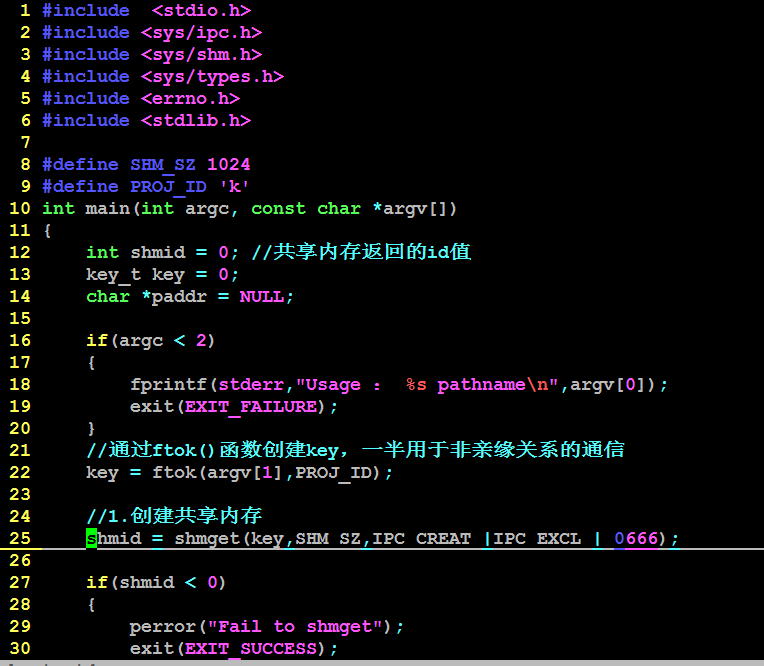
**struct ipc\_perm shm\_perm;/\* 操作权限\*/  
       int shm\_segsz;                    /\*段的大小（以字节为单位）\*/  
      time\_t shm\_atime;          /\*最后一个进程附加到该段的时间\*/  
       time\_t shm\_dtime;          /\*最后一个进程离开该段的时间\*/  
      time\_t shm\_ctime;          /\*最后一个进程修改该段的时间\*/  
      unsigned short shm\_cpid;   /\*创建该段进程的pid\*/  
       unsigned short shm\_lpid;   /\*在该段上操作的最后1个进程的pid\*/  
       short shm\_nattch;          /\*当前附加到该段的进程的个数\*/  
 /\*下面是私有的\*/  
        unsigned short shm\_npages;  /\*段的大小（以页为单位）\*/  
      unsigned long \*shm\_pages;   /\*指向frames->SHMMAX的指针数组\*/  
      struct vm\_area\_struct \*attaches; /\*对共享段的描述\*/**

**}；**

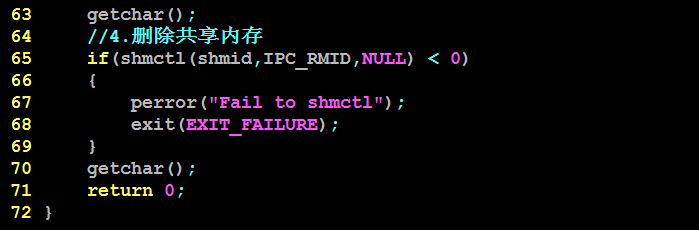
**IPC\_RMID 破化共享内存段。**



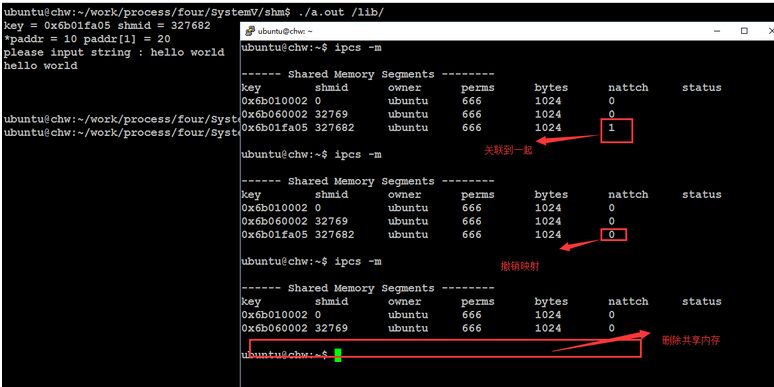
**代码演示：**







**运行结果：**



**练习:利用共享内存实现两个进程间的shm\_write.C和shm\_read.C之间的通信。**

**[注意：两个进程只要打开的是同一个文件路径，则获得相同的共享内存。]**

