Linux IPC 可参照<http://www.cnblogs.com/wangkangluo1/archive/2012/05/14/2498786.html>

http://www.cnblogs.com/andtt/articles/2136279.html

IPC方式：

信号

信号灯（注意与信号的区别）

管道

共享内存

消息队列

套接口

共享内存：http://blog.csdn.net/taomeegaoke/article/details/7493319

共享内存对象：

   数据结构shmid\_kernel的定义如下：  
    struct shmid\_kernel  
    {      
        struct shmid\_ds u;          
        unsigned long shm\_npages;   
        unsigned long \*shm\_pages;     
        struct vm\_area\_struct \*attaches;   
    };

系统中共享内存对象表

struct shmid\_kernel \*shm\_segs[SHMMNI] SHMMNI一般为128

对共享内存对象的操作：

1. 创建或者获取共享内存对象

Int Shmget(key ,size,flag)

1），Key=0时会创建一个新的对象

\* 算出size要占用的页数，检查其合法性。  
    \* 申请一块内存用于建立shmid\_kernel数据结构，注意这里申请的内存区域大小不包括真正的共享内存区，实际上，要等到第一个进程试图访问它的时候才真正创建共享内存区。  
    \* 根据该共享内存区所占用的页数，为其申请一块空间用于建立页表（每页4个字节），将页表清0。  
    \* 搜索向量表shm\_segs，为新创建的共享内存对象找一个空位置。  
    \* 填写shmid\_kernel数据结构，将其加入到向量表shm\_segs中为其找到的空位置。  
    \* 返回该共享内存对象的引用标识符。

   2) 在向量表shm\_segs中查找键值为key的共享内存对象，结果有三：  
    \* 如果没有找到，而且在操作标志shmflg中没有指明要创建新共享内存，则错误返回，否则创建一个新的共享内存对象。  
    \* 如果找到了，但该次操作要求必须创建一个键值为key的新对象，那么错误返回。  
    \* 否则，合法性、认证检查，如有错，则错误返回；否则，返回该内存对象的引用标识符。  
   
    共享内存对象的创建者可以控制对于这块内存的访问权限和它的key是公开还是私有。如果有足够的权限，它也可以把共享内存锁定在物理内存中

1. 关联： 把共享内存对象空间映射到自己的进程空间：

Void \*Shmat(id,shmaddr flg)

其中：  
     shmid是shmget返回的共享内存对象的引用标识符；  
    shmaddr用来指定该共享内存区域在进程的虚拟地址空间对应的虚拟地址；  
    shmflg是映射标志；  
    返回的是在进程中的虚拟地址  
   
    该函数所做的工作如下：  
    1) 根据shmid找到共享内存对象。  
    2) 如果shmaddr为0，即用户没有指定该共享内存区域在它的虚拟空间中的位置，则由系统在进程的虚拟地址空间中为其找一块区域（从1G开始）；否则，就用shmaddr作为映射的虚拟地址。  
  (If  shmaddr  is NULL, the system chooses a suitable (unused) address a他 which to attach the segment)  
    3) 检查虚拟地址的合法性（不能超过进程的最大虚拟空间大小—3G，不能太接近堆栈栈顶）。  
    4) 认证检查。  
    5) 申请一块内存用于建立数据结构vm\_area\_struct，填写该结构。  
    6) 检查该内存区域，将其加入到进程的mm结构和该共享内存对象的vm\_area\_struct队列中。  
   
    共享内存的粘附只是创建一个vm\_area\_struct数据结构，并将其加入到相应的队列中，此时并没有创建真正的共享内存页。  
   
    当进程第一次访问共享虚拟内存的某页时，因为所有的共享内存页还都没有分配，所以会发生一个page fault异常。当Linux处理这个page fault的时候，它找到发生异常的虚拟地址所在的vm\_area\_struct数据结构。在该数据结构中包含有这类共享虚拟内存的一组处理程序，其中的 nopage操作用来处理虚拟页对应的物理页不存在的情况。对共享内存，该操作是shm\_nopage（定义在ipc/shm.c中）。该操作在描述这个共享内存的shmid\_kernel数据结构的页表shm\_pages中查找发生page fault异常的虚拟地址所对应的页表条目，看共享页是否存在（页表条目为0，表示共享页是第一次使用）。如果不存在，它就分配一个物理页，并为它创建一个页表条目。这个条目不但进入当前进程的页表，同时也存到shmid\_kernel数据结构的页表shm\_pages中。  
   
    当下一个进程试图访问这块内存并得到一个page fault的时候，经过同样的路径，也会走到函数shm\_nopage。此时，该函数查看shmid\_kernel数据结构的页表shm\_pages时，发现共享页已经存在，它只需把这里的页表项填到进程页表的相应位置即可，而不需要重新创建物理页。所以，是第一个访问共享内存页的进程使得这一页被创建，而随后访问它的其它进程仅把此页加到它们的虚拟地址空间。

3,分离

Shmdt(char \*shmaddr)

其中shmaddr是进程要分离的共享页的开始虚拟地址。  
   
    该函数搜索进程的内存结构中的所有vm\_area\_struct数据结构，找到地址shmaddr对应的一个，调用函数do\_munmap将其释放。  
   
    在函数do\_munmap中，将要释放的vm\_area\_struct数据结构从进程的虚拟内存中摘下，清除它在进程页表中对应的页表项（可能占多个页表项）.   
    
    如果共享的虚拟内存没有被锁定在物理内存中，分离会更加复杂。因为在这种情况下，共享内存的页可能在系统大量使用内存的时候被交换到系统的交换磁盘。为了避免这种情况，可以通过下面的控制操作，将某共享内存页锁定在物理内存不允许向外交换。共享内存的换出和换入，已在第3章中讨论。

4,控制

Shmctl(id ,cmd,\*buf)

Linux在共享内存上实现的第四种操作是共享内存的控制（call值为SHMCTL的sys\_ipc调用），它由函数sys\_shmctl实现。控制操作包括获得共享内存对象的状态，设置共享内存对象的参数（如uid、gid、mode、ctime等），将共享内存对象在内存中锁定和释放（在对象的mode上增加或去除SHM\_LOCKED标志），释放共享内存对象资源等。  
   
    共享内存提供了一种快速灵活的机制，它允许进程之间直接共享大量的数据，而无须使用拷贝或系统调用。共享内存的主要局限性是它不能提供同步，如果两个进程企图修改相同的共享内存区域，由于内核不能串行化这些动作，因此写的数据可能任意地互相混合。所以使用共享内存的进程必须设计它们自己的同步协议，如用信号灯等。