共享内存是效率最高的 IPC，因为他抛弃了内核这个“代理人”，直截了当地将一块裸露的内存放在需要数据传输的进程面前，让他们自己搞，这样的代价是：这些进程必须小心谨慎地操作这块裸露的共享内存，做好诸如同步、互斥等工作，毕竟现在没有人帮他们来管理了，一切都要自己动手。也因为这个原因，共享内存一般不能单独使用，而要配合信号量、互斥锁等协调机制，让各个进程在高效交换数据的同时，不会发生数据践踏、破坏等意外。

共享内存的思想很朴素，进程与进程之间虚拟内存空间本来相互独立，不能互相访问的，但是可以通过某些方式，使得相同的一块物理内存多次映射到不同的进程虚拟空间之中，这样的效果就相当于多个进程的虚拟内存空间部分重叠在一起，看示意图：

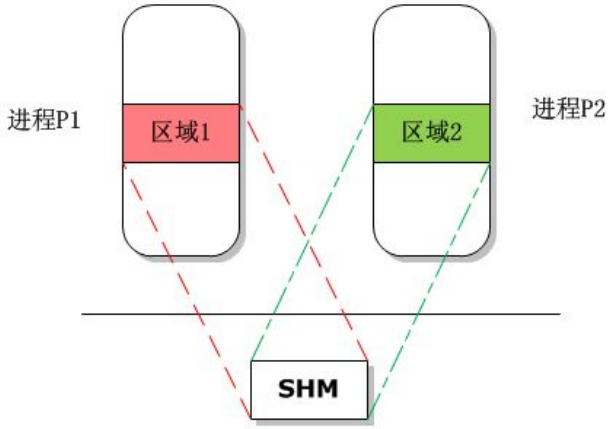

图 5-35 共享内存的逻辑

像上图所示，当进程 P1 向其虚拟内存中的区域 1 写入数据时，进程 2 就能同时在其虚拟内存空间的区域 2 看见这些数据，中间没有经过任何的转发，效率极高。

使用共享内存的一般步骤是：

1，获取共享内存对象的 ID

2，将共享内存映射至本进程虚拟内存空间的某个区域

3，当不再使用时，解除映射关系

4，当没有进程再需要这块共享内存时，删除它。

下面详细解剖共享内存（SHM，即 SHared Memory）的 API。

功能	获取共享内存的 ID		
头文件	#include <sys/ipc.h> #include <sys/shm.h>		
原型	int **shmget**(key_t key, size_t size, int shmflg);		
参数	key	共享内存的键值	
	size	共享内存的尺寸（PAGE_SIZE 的整数倍）	
	shmflg	IPC_CREAT	如果 key 对应的共享内存不存在，则创建之
		IPC_EXCL	如果该 key 对应的共享内存已存在，则报错
		SHM_HUGETLB	使用“大页面”来分配共享内存
		SHM_NORESERVE	不在交换分区中为这块共享内存保留空间
		mode	共享内存的访问权限（八进制，如 0644）
返回值	成功	该共享内存的 ID	
	失败	-1	

备注	如果 key 指定为为 IPC_PRIVATE，则会自动产生一个随机未用的新键值

图 5-36 函数 shmget( )的接口规范

所谓的“大页面”指的是内核为了提高程序性能，对内存实行分页管理时，采用比默认尺寸（4KB）更大的分页，以减少缺页中断。LINUX 内核支持以 2MB 作为物理页面分页的基本单位。

功能	对共享内存进行映射，或者解除映射		
头文件	#include <sys/types.h> #include <sys/shm.h>		
原型	void ***shmat**(int shmid, const void *shmaddr, int shmflg); int **shmdt**(const void *shmaddr);		
参数	shmid	共享内存 ID	
	shmaddr	shmat( )	1，如果为 NULL，则系统会自动选择一个合适的虚拟内存空间地址去映射共享内存。 2，如果不为 NULL，则系统会根据 shmaddr 来选择一个合适的内存区域。
		shmdt( )	共享内存的首地址
	shmflg	SHM_RDONLY	以只读方式映射共享内存
		SHM_REMAP	重新映射，此时 shmaddr 不能为 NULL
		SHM_RND	自动选择比 shmaddr 小的最大页对齐地址
返回值	成功	共享内存的首地址	
	失败	-1	
备注	无		

图 5-37 共享内存的映射和解除映射函数接口规范

1，共享内存只能以只读或者可读写方式映射，无法以只写方式映射。

2，shmat( )第二个参数 shmaddr 一般都设为 NULL，让系统自动找寻合适的地址。但当其确实不为空时，那么要求 SHM_RND 在 shmflg 必须被设置，这样的话系统将会选择比 shmaddr 小而又最大的页对齐地址（即为 SHMLBA 的整数倍）作为共享内存区域的起始地址。如果没有设置 SHM_RND，那么 shmaddr 必须是严格的页对齐地址。

总之，映射时将 shmaddr 是更明智的做法，因为他更简单，也更具移植性。

3，解除映射之后，进程不能再允许访问 SHM。

功能	获取或者设置共享内存的相关属性		
头文件	#include <sys/ipc.h> #include <sys/shm.h>		
原型	int **shmctl**(int shmid, int cmd, struct shmid_ds *buf);		
参数	shmid	共享内存 ID	
	cmd	IPC_STAT	获取属性信息，放置到 buf 中

		IPC_SET	设置属性信息为 buf 指向的内容
		IPC_RMID	将共享内存标记为“即将被删除”状态
		IPC_INFO	获得关于共享内存的系统限制值信息
		SHM_INFO	获得系统为共享内存消耗的资源信息
		SHM_STAT	同 IPC_STAT，但 shmid 为该 SHM 在内核中记录所有 SHM 信息的数组的下标，因此通过迭代所有的下标可以获得系统中所有 SHM 的相关信息
		SHM_LOCK	禁止系统将该 SHM 交换至 swap 分区
		SHM_UNLOCK	允许系统将该 SHM 交换至 swap 分区
	buf	属性信息结构体指针	
返回值	成功	IPC_INFO	内核中记录所有 SHM 信息的数组的下标最大值
		SHM_INFO	
		SHM_STAT	下标值为 shmid 的 SHM 的 ID
	失败	-1	
备注	无		

图 5-38 函数 shmctl( )的接口规范

使用以上接口需要知道的几点：

1，IPC_STAT 获得的属性信息被存放在以下结构体中：

```
struct shmid_ds
{
    struct ipc_perm shm_perm;    /* 权限相关信息 */
    size_t          shm_segsz;   /* 共享内存尺寸（字节） */
    time_t          shm_atime;   /* 最后一次映射时间 */
    time_t          shm_dtime;   /* 最后一个解除映射时间 */
    time_t          shm_ctime;   /* 最后一次状态修改时间 */
    pid_t           shm_cpid;    /* 创建者 PID */
    pid_t           shm_lpid;    /* 最后一次映射或解除映射者 PID */
    shmatt_t        shm_nattch;  /* 映射该 SHM 的进程个数 */
};
```

其中权限信息结构体如下：

```
struct ipc_perm
{
    key_t          __key;    /* 该 SHM 的键值 key */
    uid_t          uid;      /* 所有者的有效 UID */
    gid_t          gid;      /* 所有者的有效 GID */
    uid_t          cuid;     /* 创建者的有效 UID */
    gid_t          cgid;     /* 创建者的有效 GID */
    unsigned short mode;     /* 读写权限 +
                                SHM_DEST +
                                SHM_LOCKED 标记 */
    unsigned short __seq;    /* 序列号 */
};
```

2，当使用 IPC_RMID 后，上述结构体 struct ipc_perm 中的成员 mode 将可以检测出 SHM_DEST，但 SHM 并不会被真正删除，要等到 shm_nattch 等于 0 时才会被真正删除。

IPC_RMID 只是为删除做准备，而不是立即删除。

3，当使用 IPC_INFO 时，需要定义一个如下结构体来获取系统关于共享内存的限制值信息，并且将这个结构体指针强制类型转化为第三个参数的类型。

```
struct shminfo
{
    unsigned long shmmax;   /* 一块 SHM 的尺寸最大值 */
    unsigned long shmmin;   /* 一块 SHM 的尺寸最小值（永远为 1） */
    unsigned long shmmni;   /* 系统中 SHM 对象个数最大值 */
    unsigned long shmseg;   /* 一个进程能映射的 SHM 个数最大值 */
    unsigned long shmall;   /* 系统中 SHM 使用的内存页数最大值 */
};
```

4，使用选项 SHM_INFO 时，必须保证宏_GNU_SOURCE 有效。获得的相关信息被存放在如下结构体当中：

```
struct shm_info
{
    int used_ids;                   /* 当前存在的 SHM 个数 */
    unsigned long shm_tot;   /* 所有 SHM 占用的内存页总数 */
    unsigned long shm_rss;   /* 当前正在使用的 SHM 内存页个数 */
    unsigned long shm_swp;   /* 被置入交换分区的 SHM 个数 */
    unsigned long swap_attempts;     /* 已废弃 */
    unsigned long swap_successes;    /* 已废弃 */
};
```

5，注意：选项 SHM_LOCK 不是锁定读写权限，而是锁定 SHM 能否与 swap 分区发生交换。一个 SHM 被交换至 swap 分区后如果被设置了 SHM_LOCK，那么任何访问这个 SHM 的进程都将会遇到页错误。进程可以通过 IPC_STAT 后得到的 mode 坚持测到 SHM_LOCKED 信息。