第七章：共享内存

7.1 共享内存概述

共享内存允许两个或者多个进程共享给定的存储区域。

共享内存的特点

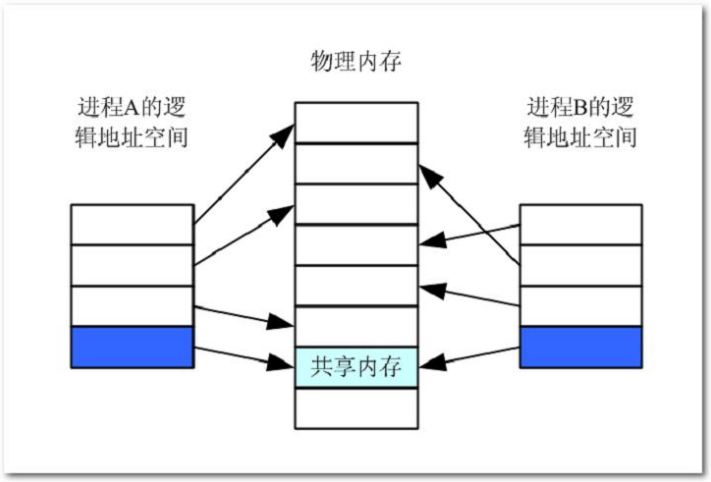
1、 共享内存是进程间共享数据的一种最快的方法。

一个进程向共享的内存区域写入了数据，共享这个内存区域的所有进程就可以立刻看到其中的内容。

2、使用共享内存要注意的是多个进程之间对一个给定存储区访问的互斥。

若一个进程正在向共享内存区写数据，则在它做完这一步操作前，别的进程不应当去读、写这些数据。

共享内存示意图



在 ubuntu 12.04 中共享内存限制值如下

1、共享存储区的最小字节数：1

2、共享存储区的最大字节数：32M

3、共享存储区的最大个数：4096

4、每个进程最多能映射的共享存储区的个数：4096

7.2 共享内存操作

7.2.1 获得一个共享存储标识符

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size,int shmflg);
```

功能：

创建或打开一块共享内存区

参数：

key：IPC 键值
size：该共享存储段的长度(字节)
shmflg：标识函数的行为及共享内存的权限。
IPC_CREAT：如果不存在就创建
IPC_EXCL：如果已经存在则返回失败
位或权限位：共享内存位或权限位后可以设置共享内存的访问权限，格式和 open 函数的 mode_t 一样，但可执行权限未使用。

返回值:

成功：返回共享内存标识符。
失败：返回－1。

使用 shell 命令操作共享内存:

查看共享内存

ipcs -m

删除共享内存

ipcrm -m shmid

7.2.2 共享内存映射(attach)

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

功能：

将一个共享内存段映射到调用进程的数据段中。

参数：

shmid：共享内存标识符。
shmaddr：共享内存映射地址(若为 NULL 则由系统自动指定)，推荐使用 NULL。
shmflg：共享内存段的访问权限和映射条件
0：共享内存具有可读可写权限。
SHM_RDONLY：只读。
SHM_RND：（shmaddr 非空时才有效）
没有指定 SHM_RND 则此段连接到 shmaddr 所指定的地址上(shmaddr 必需页对齐)。
指定了 SHM_RND 则此段连接到 shmaddr- shmaddr%SHMLBA 所表示的地址上。

返回值：

成功：返回共享内存段映射地址
失败：返回 -1

注意：

shmat 函数使用的时候第二个和第三个参数一般设为 NULL 和 0，即系统自动指定共享内存地址，并且共享内存可读可写。

7.2.3 解除共享内存映射(detach)

```
#include <sys/types.h>
```

#include <sys/shm.h>
int shmdt(const void *shmaddr);

功能：

将共享内存和当前进程分离(仅仅是断开联系并不删除共享内存)。

参数：

shmaddr：共享内存映射地址。

返回值：

成功返回 0，失败返回 -1。

7.2.4 共享内存控制

```
#include <sys/ipc.h>
#include <sys/shm.h>
    int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

功能：

共享内存空间的控制。

参数：

shmid：共享内存标识符。
cmd：函数功能的控制。
buf：shmid_ds 数据类型的地址，用来存放或修改共享内存的属性。
cmd：函数功能的控制
IPC_RMID：删除。
IPC_SET：设置 shmid_ds 参数。
IPC_STAT：保存 shmid_ds 参数。
SHM_LOCK：锁定共享内存段(超级用户)。
SHM_UNLOCK：解锁共享内存段。

返回值：

成功返回 0，失败返回 -1。

例：02_shared_memory_write.c

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

#define BUFSZ 2048

int main(int argc, char *argv[])
{
```

```
int shmid;
int ret;
key_t key;
char *shmadd;

key = ftok(".", 2012);
if(key==-1)
{
    perror("ftok");
}
system("ipcs -m");
/*打开共享内存*/
shmid = shmget(key, BUFSZ, IPC_CREAT|0666);
if(shmid < 0)
{
    perror("shmget");
    exit(-1);
}
/*映射*/
shmadd = shmat(shmid, NULL, 0);
if(shmadd < 0)
{
    perror("shmat");
    exit(-1);
}
/*读共享内存区数据*/
printf("data = [%s]\n", shmadd);
/*分离共享内存和当前进程*/
ret = shmdt(shmadd);
if(ret < 0)
{
    perror("shmdt");
    exit(1);
}
else
{
    printf("deleted shared-memory\n");
}
```

```
    /*删除共享内存*/
    shmctl(shmid, IPC_RMID, NULL);
    system("ipcs -m");
    return 0;
}
```

例：02_shared_memory_read.c

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

#define BUFSZ 2048

int main(int argc, char *argv[])
{
    int shmid;
    int ret;
    key_t key;
    char *shmadd;

    key = ftok(".", 2012);
    if(key == -1)
    {
        perror("ftok");
    }
    /*创建共享内存*/
    shmid = shmget(key, BUFSZ, IPC_CREAT|0666);
    if(shmid < 0)
    {
        perror("shmget");
        exit(-1);
    }
    /*映射*/
    shmadd = shmat(shmid, NULL, 0);
```

```
    if(shmadd < 0)
    {
        perror("shmat");
        _exit(-1);
    }
    /*拷贝数据至共享内存区*/
    printf("copy data to shared-memory\n");
    bzero(shmadd, BUFSZ);
    strcpy(shmadd, "data in shared memory\n");
    return 0;
}
```

注意：

SHM_LOCK 用于锁定内存，禁止内存交换。并不代表共享内存被锁定后禁止其它进程访问。其真正的意义是：被锁定的内存不允许被交换到虚拟内存中。

这样做的优势在于让共享内存一直处于内存中，从而提高程序性能。