## [linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "_blank)中的两种共享内存：

一种是我们的IPC通信System V版本的共享内存，另外的一种就是存储映射I/O（mmap函数）。

共享内存就是多个进程的[地址空间](https://so.csdn.net/so/search?q=%E5%9C%B0%E5%9D%80%E7%A9%BA%E9%97%B4&spm=1001.2101.3001.7020)映射到同一个物理内存，多个进程都能看到这块物理内存，共享内存可以提供给服务器进程和客户进程之间进行通信，不需要进行数据的复制，所以速度最快。为了实现更安全通信，往往还与信号灯等同步机制共同使用。

## 背景：普通文件读写

普通的读写文件的原理，进程调用read或是write后会陷入内核，因为这两个函数都是系统调用，进入系统调用后，内核开始读写文件，假设内核在读取文件，内核首先把文件读入自己的内核空间，读完之后进程在内核回归用户态，内核把读入内核内存的数据再copy进入进程的用户态内存空间。实际上我们同一份文件内容相当于读了两次，先读入内核空间，再从内核空间读入用户空间。

## mmap介绍

server端

void\* mmap(void\* addr, size\_t length, int prot, int flags, int fd, off\_t offset);

示例代码：

1. **struct** STU
2. {
3. **int** age;
4. **char** name[20];
5. **char** sex;
6. };
7. **struct** STU student = {10,"xiaoming",'m'};
8. **int** fd = open(argv[1],O\_RDWR|O\_CREAT|O\_TRUNC,0644);
9. **if**(fd < 0)
10. {
11. perror("open");
12. exit(2);
13. }
14. ftruncate(fd,**sizeof**(**struct** STU)); //文件拓展大小。
15. // lseek和ftruncate，truncate都可以达到修改文件可映射大小的结果，不过lseek可以在readonly的情况下修改，而truncate不行。事实上，lseek不改变文件实际大小，而truncate 以及 ftruncate是会改变文件实际大小的。所谓的实际大小，是指通过fstat等接口获取的文件size，不是占用的block的大小。truncate 在缩小了文件之后，如果原来seek的位置小于缩小之后的文件大小，则保持不变，如果大于，则位于文件尾，如果truncate 放大了文件，则seek位置不变。
16. //创建一个结构体大小的共享映射区。共享映射区我们可以当做数组区看待。
17. **struct** STU \*p = (**struct** STU\*)mmap(NULL,**sizeof**(**struct** STU),PROT\_READ|PROT\_WRITE,MAP\_SHARED,fd,0);
19. **if**(p == MAP\_FAILED)
20. {
21. perror("mmap");
22. exit(3);
23. }
24. close(fd); //关闭不用的文件描述符。
25. **while**(1)
26. {
27. memcpy(p,&student,**sizeof**(student));
28. student.age++;
29. sleep(1);
30. }
31. **int** ret = munmap(p,**sizeof**(student));
32. **if**(ret < 0)
33. {
34. perror("mmumap");
35. exit(4);
36. }

Client端

1. **struct** STU
2. {
3. **int** age;
4. **char** name[20];
5. **char** sex;
6. };
8. **int** fd = open(argv[1],O\_RDONLY,0644);
9. **if**(fd < 0)
10. {
11. perror("open");
12. exit(2);
13. }
15. **struct** STU student;
17. **struct** STU \*p = (**struct** STU\*)mmap(NULL,**sizeof**(**struct** STU),PROT\_READ,MAP\_SHARED,fd,0);
18. **if**(p == MAP\_FAILED)
19. {
20. perror("mmap");
21. exit(3);
22. }
23. close(fd);
24. **int** i = 0;
25. **while**(1)
26. {
28. printf("id = %d\tname = %s\t%c\n",p->age,p->name,p->sex);
29. sleep(2);
30. }
31. **int** ret = munmap(p,**sizeof**(student));
32. **if**(ret < 0)
33. {
34. perror("mmumap");
35. exit(4);
36. }

## Shm调用介绍

（1）我们需要利用ftok函数生成key标识符。key\_t ftok(const char \*pathname,int proj\_id).

（2）通过int shmget(key\_t key, size\_t size, int shmflg);在物理内存创建一个共享内存，返回共享内存的编号。

（3）通过void \*shmat(int shmid, constvoid shmaddr,int shmflg);连接成功后把共享内存区对象映射到调用进程的地址空间

（4）通过void \*shmdt(constvoid\* shmaddr);断开用户级页表到共享内存的那根箭头。

（5）通过int shmctl(int shmid, int cmd, struct shmid\_ds\* buf);释放物理内存中的那块共享内存。

## 总结mmap和shm:

1、mmap是在磁盘上建立一个文件，每个进程地址空间中开辟出一块空间进行映射。

而对于shm而言，shm每个进程最终会映射到同一块物理内存。shm保存在物理内存，这样读写的速度要比磁盘要快，但是存储量不是特别大。二者本质上是类似的，mmap可以看到文件的实体，而 shmget 对应的文件在交换分区上的 shm 文件系统内，无法直接 cat 查看

2、相对于shm来说，mmap更加简单，调用更加方便，所以这也是大家都喜欢用的原因。

3、进程挂了重启不丢失内容，二者都可以做到。当机器重启mmap 可以不丢失内容(文件内保存了OS同步过的映像)，而 shmget 会丢失，因为mmap把文件保存在磁盘上，这个文件还保存了操作系统同步的映像，所以mmap不会丢失，但是shmget就会丢失。

4、易用性：mmap 的接口会简单一些

5、通用性：posix 的 mmap 会相对广泛一些

6、mmap保存到实际硬盘，实际存储并没有反映到主存上。优点：储存量可以很大(多于主存)(这里一个问题，需要高手解答,会不会太多拷贝到主存里面？？？)；缺点：进程间读取和写入速度要比主存的要慢。

7、shm保存到物理存储器(主存)，实际的储存量直接反映到主存上。优点，进程间访问速度(读写)比磁盘要快；缺点，储存量不能非常大(多于主存)。使用上看：如果分配的存储量不大，那么使用shm；如果存储量大，那么使用mmap。

8、mmap系统调用并不是完全为了用于共享内存而设计的。它本身提供了不同于一般对普通文件的访问方式，进程可以像读写内存一样对普通文件的操作。而Posix或系统V的共享内存IPC则纯粹用于共享目的，当然mmap()实现共享内存也是其主要应用之一。

9、用特殊文件提供匿名内存映射：适用于具有亲缘关系的进程之间；由于父子进程特殊的亲缘关系，在父进程中先调用mmap()，然后调用fork()。那么在调用fork()之后，子进程继承父进程匿名映射后的地址空间，同样也继承mmap()返回的地址，这样，父子进程就可以通过映射区域进行通信了。

10、copy onwrite

写入时复制（Copy-on-write，COW）是一种计算机程序设计领域的优化策略。其核心思想是，如果有多个调用者（callers）同时要求相同资源（如内存或磁盘上的数据存储），他们会共同获取相同的指针指向相同的资源，直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本（private copy）给该调用者，而其他调用者所见到的最初的资源仍然保持不变。这过程对其他的调用者都是透明的（transparently）。改完提交以后其他caller才会看到新修改的内容。此作法主要的优点是如果调用者没有修改该资源，就不会有副本（private copy）被创建，因此多个调用者只是读取操作时可以共享同一份资源。

11、top res shr

Top命令看进程占用的各种内存，res-shr就是实际的内存

VIRT：virtual memory usage 虚拟内存

1、进程“需要的”虚拟内存大小，包括进程使用的库、代码、数据等

2、假如进程申请100m的内存，但实际只使用了10m，那么它会增长100m，而不是实际的使用量

RES：resident memory usage 常驻内存（或者叫驻留内存（ Resident Memory））

1、进程当前使用的内存大小，但不包括swap out

2、包含其他进程的共享

3、如果申请100m的内存，实际使用10m，它只增长10m，与VIRT相反

4、关于库占用内存的情况，它只统计加载的库文件所占内存大小

SHR：shared memory 共享内存

1、除了自身进程的共享内存，也包括其他进程的共享内存

2、虽然进程只使用了几个共享库的函数，但它包含了整个共享库的大小

3、计算某个进程所独占的物理内存大小公式：RES – SHR

4、swap out后，它将会降下来

138环境设置的MDB配置参数DBSize=4096MB 用top看shr字段也是4.03GB 基本符合。

## 参考文献：

[(15条消息) mmap映射区和shm共享内存的区别总结\_hj605635529的博客-CSDN博客\_linux共享内存 mmap](https://blog.csdn.net/hj605635529/article/details/73163513)

<http://blog.csdn.net/hj605635529/article/details/67636526>

[shmget物理内存\_共享内存 mmap shmget 区别\_weixin\_39821330的博客-CSDN博客](https://blog.csdn.net/weixin_39821330/article/details/111540323)