Posix 多线程编程学习笔记（六）—共享内存（1）

一、什么是共享内存区

共享内存区是最快的可用 IPC 形式。它允许多个不相关的进程去访问同一部分逻辑内存。如果需要在两个运行中的进程之间传输数据，共享内存将是一种效率极高的解决方案。一旦这样的内存区映射到共享它的进程的地址空间，这些进程间数据的传输就不再涉及内核。这样就可以减少系统调用时间，提高程序效率。

共享内存是由 IPC 为一个进程创建的一个特殊的地址范围，它将出现在进程的地址空间中。其他进程可以把同一段共享内存段“连接到”它们自己的地址空间里去。所有进程都可以访问共享内存中的地址。如果一个进程向这段共享内存写了数据，所做的改动会立刻被有访问同一段共享内存的其他进程看到。

要注意的是共享内存本身没有提供任何同步功能。也就是说，在第一个进程结束对共享内存的写操作之前，并没有什么自动功能能够预防第二个进程开始对它进行读操作。共享内存的访问同步问题必须由程序员负责。可选的同步方式有互斥锁、条件变量、读写锁、纪录锁、信号灯。

二、mmap

在将共享内存前我们要先来介绍下面几个函数。

mmap 函数把一个文件或一个 Posix 共享内存区对象映射到调用进程的地址空间。使用该函数有三个目的：

1.使用普通文件以提供内存映射 I/O

2.使用特殊文件以提供匿名内存映射。

3.使用 shm_open 以提供无亲缘关系进程间的 Posix 共享内存区。

1.

名称:：	mmap
功能：	把 I/O 文件映射到一个存储区域中
头文件：	#include <sys/mman.h>
函数原形：	void *mmap(void *addr,size_t len,int prot,int flag,int filedes,off_t off);
参数：	addr 指向映射存储区的起始地址 len 映射的字节 prot 对映射存储区的保护要求 flag flag 标志位 filedes 要被映射文件的描述符 off 要映射字节在文件中的起始偏移量
返回值：	若成功则返回映射区的起始地址，若出错则返回 MAP_FAILED

addr 参数用于指定映射存储区的起始地址。通常将其设置为 NULL，这表示由系统选择该映射区的起始地址。

filedes 指要被映射文件的描述符。在映射该文件到一个地址空间之前，先要打开该文件。len 是映射的字节数。

off 是要映射字节在文件中的起始偏移量。通常将其设置为 0。

prot 参数说明对映射存储区的保护要求。可将 prot 参数指定为 PROT_NONE,或者是 PROT_READ(映射区可读),PROT_WRITE（映射区可写）,PROT_EXEC（映射区可执行）任意组合的按位或，也可以是 PROT_NONE(映射区不可访问)。对指定映射存储区的保护要求不能超过文件 open 模式访问权限。

flag 参数影响映射区的多种属性：

MAP_FIXED 返回值必须等于 addr.因为这不利于可移植性，所以不鼓励使用此标志。

MAP_SHARED 这一标志说明了本进程对映射区所进行的存储操作的配置。此标志指定存储操作修改映射文件。

MAP_PRIVATE 本标志导致对映射区建立一个该映射文件的一个私有副本。所有后来对该映射区的引用都是引用该副本，而不是原始文件。

要注意的是必须指定 MAP_FIXED 或 MAP_PRIVATE 标志其中的一个，指定前者是对存储映射文件本身的一个操作，而后者是对其副本进行操作。

mmap 成功返回后，fd 参数可以关闭。该操作对于由 mmap 建立的映射关系没有影响。为从某个进程的地址空间删除一个映射关系，我们调用 munmap.

2.

名称:：	munmap
功能：	解除存储映射
头文件：	#include <sys/mman.h>
函数原形：	int munmap(caddr_t addr,size_t len);
参数：	addr 指向映射存储区的起始地址 len 映射的字节
返回值：	若成功则返回 0，若出错则返回-1

其中 addr 参数是由 mmap 返回的地址，len 是映射区的大小。再次访问这些地址导致向调用进程产生一个 SIGSEGV 信号。

如果被映射区是使用 MAP_PRIVATE 标志映射的，那么调用进程对它所作的变动都被丢弃掉。

内核的虚存算法保持内存映射文件（一般在硬盘上）与内存映射区（在内存中）的同步（前提它是 MAP_SHARED 内存区）。这就是说，如果我们修改了内存映射到某个文件的内存区中某个位置的内容，那么内核将在稍后某个时刻相应地更新文件。然而有时候我们希望确信硬盘上的文件内容与内存映射区中的文件内容一致，于是调用 msync 来执行这种同步。

3.

名称:：	msync
功能：	同步文件到存储器
头文件：	#include <sys/mman.h>
函数原形：	int msync(void *addr,size_t len,int flags);
参数：	addr 指向映射存储区的起始地址 len 映射的字节 prot flags
返回值：	若成功则返回 0，若出错则返回-1

其中 addr 和 len 参数通常指代内存中的整个内存映射区，不过也可以指定该内存区的一个子集。flags 参数为 MS_ASYNC(执行异步写)，MS_SYNC（执行同步写），MS_INVALIDATE（使高速缓存的数据实效）。其中 MS_ASYNC 和 MS_SYNC 这两个常值中必须指定一个，但不能都指定。它们的差别是，一旦写操作已由内核排入队列，MS_ASYNC 即返回，而 MS_SYNC 则要等到写操作完成后才返回。如果还指定了 MS_INVALIDATE，那么与其最终拷贝不一致的文件数据的所有内存中拷贝都失效。后续的引用将从文件取得数据。

4.

名称:：	memcpy
功能：	复制映射存储区
头文件：	#include <string.h>
函数原形：	void *memcpy(void *dest,const void *src,size_t n);
参数：	dest 待复制的映射存储区 src 复制后的映射存储区 n 待复制的映射存储区的大小
返回值：	返回 dest 的首地址

memcpy 拷贝 n 个字节从 dest 到 src。

下面就是利用 mmap 函数影射 I/O 实现的 cp 命令。

```
/*mycp.c*/
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc,char *argv[])
{
    int fdin,fdout;
    void *arc,dst;
    struct stat statbuf;
```

```
    if(argc!=3)
    {
        printf("please input two file!\n");
        exit(1);
    }
    if((fdin=open(argv[1],O_RDONLY))<0) /*打开原文件*/
        perror(argv[1]);
    if((fdout=open(argv[2],O_RDWR|O_CREAT|O_TRUNC))<0)/*创建并打开目标文件*/
        perror(argv[2]);

    if(fstat(fdin,&statbuf)<0) /*获得文件大小信息*/
        printf("fstat error");

    if(lseek(fdout,statbuf.st_size-1,SEEK_SET)==-1)/*初始化输出映射存储区*/
        printf("lseek error");
    if(write(fdout,"1")!=1)
        printf("write error");

    if((src=mmap(0,statbuf.st_size,PROT_READ,MAP_SHARED,fdin,0))==MAP_FAILED)
        /*映射原文件到输入的映射存储区*/
        printf("mmap error);
    if((dst=mmap(0,statbuf.st_size,PROT_READ|PROT_WRITE,MAP_SHARED,fdout,0))
==MAP_FAILED) /*映射目标文件到输出的映射存储区*/
        printf("mmap error);
    memcpy(dst,src,statbuf.st_size);/*复制映射存储区*/
    munmap(src,statbuf.st_size); /*解除输入映射*/
    munmap(dst,statbuf.st_size); /*解除输出映射*/
    close(fdin);
    close(fdout);
}
```

下面是运行结果：

```
#cc –o mycp mycp.c
#./mycp test1 test2
```

Posix 多线程编程学习笔记（六）—共享内存（2）

三、posix 共享内存函数

posix 共享内存区涉及两个步骤：

1、指定一个名字参数调用 shm_open,以创建一个新的共享内存区对象或打开一个以存在的共享内存区对象。

2、调用 mmap 把这个共享内存区映射到调用进程的地址空间。传递给 shm_open 的名字参数随后由希望共享该内存区的任何其他进程使用。

5.

名称:：	shm_open
功能：	打开或创建一个共享内存区
头文件：	#include <sys/mman.h>
函数原形：	int shm_open(const char *name,int oflag,mode_t mode);
参数：	name　　共享内存区的名字 cflag　　标志位 mode　　权限位
返回值：	成功返回 0，出错返回-1

oflag 参数必须含有 O_RDONLY 和 O_RDWR 标志，还可以指定如下标志：O_CREAT,O_EXCL 或 O_TRUNC.

mode 参数指定权限位，它指定 O_CREAT 标志的前提下使用。

shm_open 的返回值是一个整数描述字，它随后用作 mmap 的第五个参数。

6.

名称:：	shm_unlink
功能：	删除一个共享内存区
头文件：	#include <sys/mman.h>
函数原形：	int shm_unlink(const char *name);
参数：	name　　共享内存区的名字
返回值：	成功返回 0，出错返回-1

shm_unlink 函数删除一个共享内存区对象的名字，删除一个名字仅仅防止后续的 open,mq_open 或 sem_open 调用取得成功。

下面是创建一个共享内存区的例子：

```
/*shm_open.c 创建共享内存区*/
#include <sys/mman.h>
#include <stdio.h>
#include <fcntl.h>

int main(int argc,char **argv)
{
    int shm_id;
```

```
    if(argc!=2)
    {
        printf("usage:shm_open <pathname>\n");
        exit(1);
    }
    shm_id=shm_open(argv[1],O_RDWR|O_CREAT,0644);
    printf("shmid:%d\n",shm_id);
    shm_unlink(argv[1]);
}
```

下面是运行结果，注意编译程序我们要加上“-lrt”参数。

```
#cc –lrt –o shm_open shm_open.c
#./shm_open test
shm_id:3
```

四、ftruncate 和 fstat 函数

普通文件或共享内存区对象的大小都可以通过调用 ftruncate 修改。

7.

名称:：	ftruncate
功能：	调整文件或共享内存区大小
头文件：	#include <unistd.h>
函数原形：	int ftruncate(int fd,off_t length);
参数：	fd 描述符 length 大小
返回值：	成功返回 0，出错返回-1

当打开一个已存在的共享内存区对象时，我们可调用 fstat 来获取有关该对象的信息。

8.

名称:：	fstat
功能：	获得文件或共享内存区的信息
头文件：	#include <unistd.h> #include <sys/types.h> #include <sys/stat.h>
函数原形：	int stat(const char *file_name,struct stat *buf);
参数：	file_name 文件名 buf stat 结构
返回值：	成功返回 0，出错返回-1

对于普通文件 stat 结构可以获得 12 个以上的成员信息，然而当 fd 指代一个共享内存区对象时，只有四个成员含有信息。

```
struct stat{
    mode_t st_mode;
    uid_t st_uid;
    gid_t st_gid;
    off_t st_size;
    };
```

```
/*shm_show.c 显示共享区信息*/
#include <unistd.h>
#include <sys/type.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc,char **argv)
{
     int shm_id;
    struct stat buf;

    if(argc!=2)
    {
        printf("usage:shm_open <pathname>\n");
        exit(1);
    }
    shm_id=shm_open(argv[1],O_RDWR|O_CREAT,0644);/*创建共享内存*/
    ftruncate(shm_id,100);/*修改共享内存的打开*/
    fstat(shm_id,&buf); /*把共享内存的信息记录到 buf 中*/
    printf("uid_t:%d\n",buf.st_uid); /*共享内存区所有者 ID*/
    printf("git_t:%d\n",buf.st_gid); /*共享内存区所有者组 ID*/
    printf("size :%d\n",buf.st_size); /*共享内存区大小*/
}
```

下面是运行结果：

```
#cc –lrt –o shm_show shm_show.c
#./shm_show test
uid_t:0
git_t:0
size:100
```

Posix 多线程编程学习笔记（六）—共享内存（3）

五、共享内存区的写入和读出

上面我们介绍了 mmap 函数，下面我们就可以通过这些函数，把进程映射到共享内存区。然后我们就可以通过共享内存区进行进程间通信了。

下面是共享内存区写入的例子：

```
/*shm_write.h 写入/读出共享内存区*/
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

int main(int argc,char **argv)
{
    int shm_id;
    struct stat buf;
    char *ptr;

    if(argc!=2)
    {
        printf("usage:shm_open <pathname>\n");
        exit(1);
    }
    shm_id=shm_open(argv[1],O_RDWR|O_CREAT,0644);/*创建共享内存区*/
    ftruncate(shm_id,100);/*修改共享区大小*/
    fstat(shm_id,&buf);
    ptr=mmap(NULL,buf.st_size,PROT_READ|PROT_WRITE,MAP_SHARED,shm_id,0);/*
连接共享内存区*/
    strcpy(ptr,"hello linux");/*写入共享内存区*/
    printf("%s\n",ptr);/*读出共享内存区*/
    shm_unlink(argv[1]);/*删除共享内存区*/
}
```

下面是运行结果：

```
#cc –lrt –o shm_write shm_write.c
#./shm_write test
hello linux
```

六、程序例子

下面是利用 pisix 共享内存区实现进程间通信的例子：服务器进程读出共享内存区内容，

然后清空。客户进程向共享内存区写入数据。直到用户输入“q”程序结束。程序用 posix 信号量实现互斥。

```
/*server.c 服务器程序*/
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <semaphore.h>

int main(int argc,char **argv)
{
    int shm_id;
    char *ptr;
    sem_t *sem;

    if(argc!=2)
    {
        printf("usage:shm_open <pathname>\n");
        exit(1);
    }
    shm_id=shm_open(argv[1],O_RDWR|O_CREAT,0644);/*创建共享内存区*/
    ftruncate(shm_id,100);/*调整共享内存区大小*/
    sem=sem_open(argv[1],O_CREAD,0644,1);/*创建信号量*/
    ptr=mmap(NULL,100,PROT_READ|PROT_WRITE,MAP_SHARED,shm_id,0);/*
连接共享内存区*/
    strcpy(ptr,"\0");
    while(1)
    {
        if((strcmp(ptr,"\0"))==0)/*如果为空，则等待*/
            continue;
        else
        {
            if((strcmp(ptr,"q\n"))==0)/*如果内存为 q\n 退出循环*/
                break;
            sem_wait(sem);/*申请信号量*/
            printf("server:%s",ptr);/*输入共享内存区内容*/
            strcpy(ptr,"\0");/*清空共享内存区*/
            sem_pose(sem);/*释放信号量*/
        }
        sem_unlink(argv[1]);/*删除信号量*/
        shm_unlink(argv[1]);/*删除共享内存区*/
    }
}
```

```
/*server.c 服务器程序*/
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <semaphore.h>
#include <stdio.h>

int main(int argc,char **argv)
{
    int shm_id;
    char *ptr;
    sem_t *sem;

    if(argc!=2)
    {
        printf("usage:shm_open <pathname>\n");
        exit(1);
    }
    shm_id=shm_open(argv[1],0);/*打开共享内存区
    sem=sem_open(argv[1],0);/*打开信号量*/
    ptr=mmap(NULL,100,PROT_READ|PROT_WRITE,MAP_SHARED,shm_id,0);/*
连接共享内存区*/
    while(1)
    {
        sem_wait(sem);/*申请信号量*/
        fgets(ptr,10,stdin);/*从键盘读入数据到共享内存区*/
        printf("user:%s",ptr);
        if((strcmp(ptr,"q\n"))==0)
            exit(0);
        sem_pose(sem);/*释放信号量*/
        sleep(1);
      }
      exit(0);
}
```

```
#cc –lrt –o server server.c
#cc –lrt –o user user.c
#./server test&
#./user test
输入:abc
user:abc
server:abc
```

输入:123
user:123
server:123
输入:q
user:q