一、共享内存的简介

参考文章：https://www.cnblogs.com/charlesblc/p/6261469.html

顾名思义，共享内存就是允许两个不相关的进程访问同一个逻辑内存。而如果某个进程向共享内存写入数据，所做的改动将立即影响到可以访问同一段共享内存的任何其他进程。

重要的提示：共享内存自身没有提供同步机制，需要使用信号量管理。

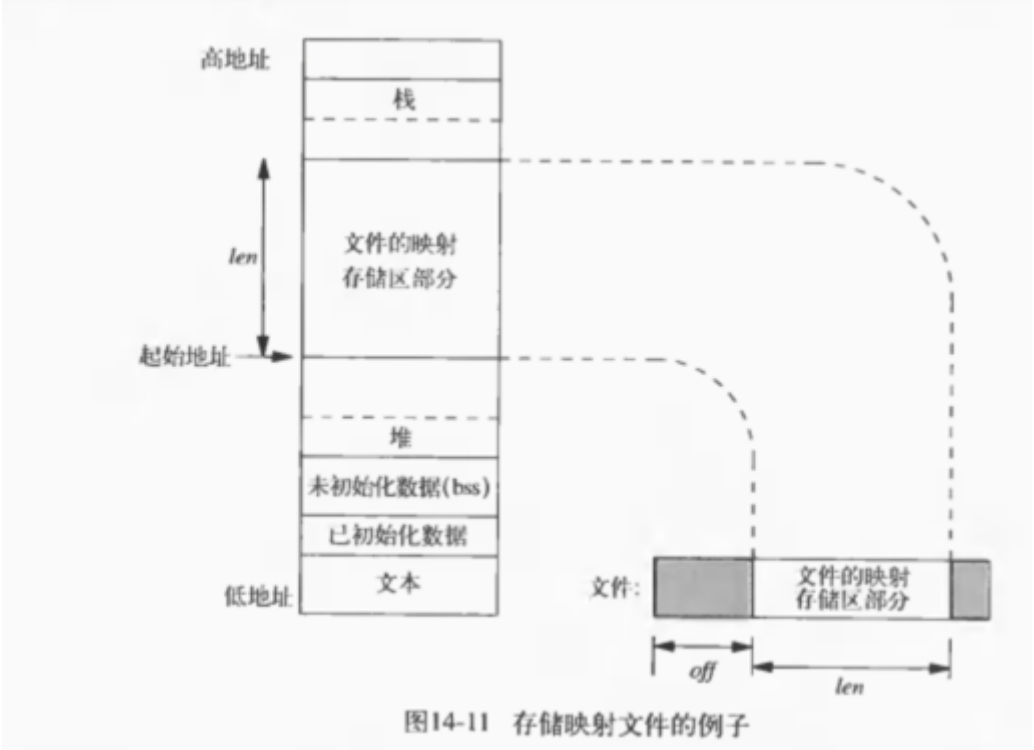

图14-11 存储映射文件的例子

二、共享内存的两套基本的API函数

2.1 代码模板

代码里并么有使用信号量进行同步管理，自行脑补。

https://github.com/yuexiuya/shm_demo

2.2 POSIX 共享内存的API

基本步骤

1) 函数shm_open和shm_unlink非常类似于为普通文件所提供的open和unlink系统调用.

2) 如果要编写一个可移植的程序,那么shm_open和shm_unlink是最好的选择.

3) shm_open:创建一个新的共享区域或者附加在已有的共享区域上.区域被其名字标识,函数返回各文件的描述符.

4) shm_unlink:类似于unlink系统调用对文件进行操作,直到所有的进程不再引用该内存区后才对其进行释放.

5) mmap:用于将一个文件映射到某一内存区中,其中也使用了shm_open函数返回的文件描述符.

6) munmap:用于释放mmap所映射的内存区域.

7) msync:同步存取一个映射区域并将高速缓存的数据回写到物理内存中,以便其他进程可以监听这些改变.

分析手段

1) 程序执行shm_open函数创建了共享内存区域,此时会在/dev/shm/创建mymem文件.

2) 通过ftruncate函数改变shm_open创建共享内存的大小为页大小(sysconf(_SC_PAGE_SIZE)),如果不执行ftruncate函数的话,会报Bus error的错误. (其实大小指定成多少都可以，1024也行，2048也行（page size的倍数？），但是一定要用ftruncate来将文件改成指定的大小，后面mmap要用的)

3) 通过mmap函数将创建的mymem文件映射到内存.

4) 通过fork派生出子进程,而共享区域映射通过fork调用而被继承.

5) 程序通过wait系统调用来保持父进程与子进程的同步.

6) 在非父子进程也可以通过共享内存区域的方式进行通讯.

2.3 System V共享内存API

基本步骤

1) System V API广泛应用于X windows系统及其扩展版本中,许多X应用程序也使用它.

2) shmget:创建一个新的共享区域或者附加在已有的共享区域上(同shm_open).

3) shmat:用于将一个文件映射到内存区域中(同mmap).

4) shmdt:用于释放所映射的内存区域(同munmap)

5) shmctl:对于多个用户,断开其对共享区域的连接(同shm_unlink)

分析手段

ipcs

这里可以 通过 ipcs 查看共享内存的分配情况，具体可参照 ipcs 的使用手册

附录 常用API的介绍

- **shmget()**

int shmget(key_t key, size_t size, int shmflg);	
头文件	#include <sys/mman.h>
描述	该函数用来创建共享内存
key	与信号量的semget函数一样，程序需要提供一个参数key（非0整数），效地为共享内存段命名. 不相关的进程可以通过该函数的返回值访问同一共享内存，它代表程序可能要的某个资源，程序对所有共享内存的访问都是间接的，程序先通过调用shmget()函供一个键，再由系统生成一个相应的共享内存标识符（shmget()函数的返回值），shmget()函数才直接使用信号量键，所有其他的信号量函数使用由semget函数返回号量标识符
size	size以字节为单位指定需要共享的内存容量
shmflg	shmflg是权限标志，它的作用与open函数的mode参数一样，如果要想标识的共享内存不存在时，创建它的话，可以与IPC_CREAT做或操作。共享内权限标志与文件的读写权限一样，举例来说，0644,它表示允许一个进程创建内存被内存创建者所拥有的进程向共享内存读取和写入数据，同时其他用户创进程只能读取共享内
return	shmget()函数成功时返回一个与key相关的共享内存标识符（非负整数），用续的共享内存函数。调用失败返回-1.

- **shmat()**

void *shmat(int shm_id, const void *shm_addr, int shmflg);	
头文件	#include <sys/mman.h>
描述	第一次创建完共享内存时，它还不能被任何进程访问，shmat()函数的作用就是 启动对该共享内存的访问，并把共享内存连接到当前进程的地址空间
shm_id	shm_id是由shmget()函数返回的共享内存标识。
shm_addr	指定共享内存连接到当前进程中的地址位置，通常为空，表示让系统来选择共享 存的地址
shm_flg	shm_flg是一组标志位，通常为0。
return	调用成功时返回一个指向共享内存第一个字节的指针，如果调用失败返回-1.

- **shmdt()**

int shmdt(const void *shmaddr);	
头文件	#include <sys/mman.h>
描述	该函数用于将共享内存从当前进程中分离。注意，将共享内存分离并不是删除它 只是使该共享内存对当前进程不再可用。
shmaddr	参数shmaddr是shmat()函数返回的地址指针
return	调用成功时返回0，失败时返回-1.

- **shmctl()**

int shmctl(int shm_id, int command, struct shmid_ds *buf);	
头文件	#include <sys/mman.h>
描述	与信号量的semctl()函数一样，用来控制共享内存
shm_id	shm_id是shmget()函数返回的共享内存标识符。
command	● IPC_STAT：把shmid_ds结构中的数据设置为共享内存的当前关联值，即用共 存的当前关联值覆盖shmid_ds的值。 ● IPC_SET：如果进程有足够的权限，就把共享内存的当前关联值设置为shmid 结构中给出的值

	● IPC_RMID：删除共享内存段

buf	buf是一个结构指针，它指向共享内存模式和访问权限的结构。
	struct shmid_ds
	{
	uid_t shm_perm.uid;
	uid_t shm_perm.gid;
	mode_t shm_perm.mode;
	};

return	调用成功时返回0，失败时返回-1.

- **shm_open()**

extern int shm_open (const char *__name, int __oflag, mode_t __mode);	
头文件	#include <sys/mman.h> -lrt
name	共享内存区的名字
oflag	标志位，O_RDONLY,O_RDWR,O_CREAT,O_EXCL,O_TRUNC (默认就选O_RDWR|O_CREAT)
mode	权限，0644，0777 之类的，控制访问群体的权限
return	成功返回0，出错返回-1

- **ftruncate ()**

int ftruncate (int __fd, __off64_t __length)	
头文件	#include <unistd.h>
描述	ftruncate()会将参数fd指定的文件大小改为参数length指定的大小。参数fd为开的文件描述词，而且必须是以写入模式打开的文件。如果原来的文件件大小数length大，则超过的部分会被删去
fd	描述符
length	指定改变后的大小
return	成功返回0，失败返回>0

- **mmap()**

void *mmap(void *addr,size_t len,int prot,int flags,int filedes,off_t off);	
头文件	#include <sys/mman.h> -lrt
描述	内存映射mmap函数的作用是建立一段可以被两个或者多个程度读写的内存段个程序对他进行任何修改，对其它程序可见
addr	用来请求使用某个特定的内存地址。如果他取0，结果指针就将自动分配（这是的做法）。否则会降低程序的可移植性，因为不同系统的可用地址范围不一样.
len	可以控制访问的数量，即内存段的数量
port	用于设置内存段访问的权限 PROT_READ|PROT_WRITE|PROT_EXEC|PROT_EXEC|PROT_NONE 读、写、行、禁止
flags	控制程序对内存段的改变所造成的影响。 //　　MAP_PRIVATE:内存段私有，对它的修改值对本进程有效 //　　MAP_SHARED:把对该内存段的修改保存到磁盘文件中 //　　MAP_FIXED:该内存段必须位于addr指定的地址处
filedes	打开的文件描述符
off	用以改变经共享内存段访问的文件中数据的起始偏移值
return	On success, mmap() returns a pointer to the mapped area. On error, the value MAP_FAILED (that is, (void *) -1) is returned, and errno is set appropriately.