# 实验04 进程的同步互斥实验

## 实验目的

1、进一步认识并发执行的实质

2、分析进程竞争资源的现象，学习解决进程同步互斥的方法

## 实验内容

1、编写程序，使用相关函数实现父子进程对共享文件的同步互斥访问。

2、修改程序，观察对临界资源访问的互斥控制的作用。

## 实验基础

### 一、临界资源的互斥访问

为了实现多进程对临界资源的互斥访问，一个进程访问临界资源的典型程序段类似如下形式：

{ ……….

**进入区**

临界区;

 **退出区**

其余代码;

………}

其中，进入区中判断资源是否可用，如果可用，则资源数量减1，进程进入临界区；否则进程阻塞等待。退出区中资源数量加1，唤醒阻塞等待该资源的进程。进入区和退出区都是原子操作。

操作系统中，通常用信号量来实现进入区和退出区，即P操作和V操作。为了实现用户程序中对某些资源的同步互斥访问，操作系统也提供了一些函数接口，功能类似于对特定临界区的进入区和退出区功能。

### 二、相关的系统调用

**（1）lockf(files，function，size) ：**用作锁定文件的某些段或者整个文件。

函数原型：

#include <unistd.h>

int lockf(int files，int function，long size)

其中：files是文件描述符；参数function可以取以下值：F\_LOCK：锁定一个区域。F\_ULOCK：解除锁定。参数size指明了从文件当前位置开始的一段连续锁定区域的长度，当size为0时，锁定记录将由当前位置一直扩展到文件尾。

如果lockf的参数function取值为F\_LOCK，而指定文件的对应区域已被其他进程锁定，那么lockf的调用进程将被阻塞直到该区域解锁。

通过使用lockf函数，可实现多进程对共享文件进行互斥访问。进程的实现中，必须使得每个进程在使用文件前对文件加锁，使用文件后解锁。

**（2）open：打开一个文件**

函数原型：#include <sys/types.h>

       #include <sys/stat.h>

       #include <fcntl.h>

       int open(char \* path，int flags，mode\_t mode);

其中：参数path 是指向所要打开的文件的路径名指针。

参数flags 规定如何打开该文件，它必须包含以下值之一 ：O\_RDONLY，只读打开 ；O\_WRONLY，只写打开 ；O\_RDWR，读/写打开；O\_CREAT，当文件不存在时创建文件，需参数mode；O\_APPEND，不论当前文件指针位置在何处，都会将文件指针移至文件尾，为write添加数据到文件；O\_TRUNC，当以可写的方式成功打开普通文件时，截断该文件的长度为0（即清空原有文件内容）。

参数mode 规定对该文件的访问权限。

open系统调用可以只使用前面介绍的这两个参数，省略第三个参数mode。第三个参数是在用O\_CREAT创建文件时使用，指出新建文件的存取许可权。由这个参数指出的存取许可权还要和umask进行运算后才得到新建文件的真正存取许可权。该运算是由umask按位取反，再按位与上第三个参数给出的数取或(~umask&mode)。例如：umask为022，mode为0770，则新建文件的存取许可权为0750即-rwxr-x---。

**（3）read：读文件**

函数原型：#include <unistd.h>

int read(int fd，void \*buf，size\_t nbytes)

    该系统调用从文件描述符fd所代表的文件中读取nbytes 个字节，到buf指定的缓冲区内。所读取的内容从当前的读/写指针所指示的位置开始，这个位置由相应的打开文件描述中的偏移值（off\_set）给出，调用成功后文件读写指针增加实际读取的字节数。

使用read 系统调用时，应注意设置的数据缓冲区充分大，能够存放所要求的数据字节，因为内核只复制数据，不进行检查。

返回：  -1：错误；0：文件偏移值是在文件结束处；整数：从该文件复制到规定的缓冲区中的字节数。通常这个字节数与所请求的字节数相同。除非请求的字节数超过剩余的字节数，这时将返回一个小于请求的字节数的数字。

**（4）write：写文件**

函数原型： #include <unistd.h>

         int write(int fd，void \*buf，size\_t nbytes)

该调用从buf所指的缓冲区中将nbytes 个字节写到描述符fd所指的文件中。

**(5)lseek：定位一个已打开文件。**

函数原型：#include <unistd.h>

int lseek(int files，off\_t offset，int whence);

系统调用根据whence指定的位置将文件描述符files指向文件的文件指针偏移offset长度的字节数。Whence的取值及其含义如下：

·SEEK\_SET：从文件头开始计算偏移量，文件指针值就是offset的值。

·SEEK\_CUR：从文件指针的当前位置开始计算偏移量，文件指针值是当前指针的值加上offset的值。

·SEEK\_END：从文件末尾开始计算偏移量，文件指针的值是文件长度加上offset的值，一般可能使用负的偏移量，使得文件指针从后向前移动。

当lseek调用成功时，返回值为一个字节为单位从文件头开始计算文件偏移量的值。调用失败时，返回值为-1。

文件指针又称文件读/写指针。文件刚打开时，文件指针指向开头位置；文件读写都是从文件指针处开始，并且在读写过程中同时移动文件指针。Lseek函数就是用于设置文件指针位置。

**(6)close：关闭文件**

函数原型：#include <unistd.h> int close(int fd);

每打开一个文件，系统就给文件分配一个文件描述符，同时为打开文件描述符的引用计数加１。Linux文件系统最多可以分配255个文件描述符。当调用close()时，打开文件描述符的引用计数值减１，最后一次对close()的调用将使应用计数值为零。

虽然当一个进程结束时，任何打开的文件将自动关闭，明显地关闭任何打开的文件是良好的程序设计习惯。

## 实验指导

1、（1）参照参考程序1，编写程序。父进程和两个子进程分别连续向共享文件中写入3行字符串。多次运行程序，观察共享文件内容。

（2）修改程序，去掉所有lock/unlock调用。多次运行程序，观察分析共享文件中，3个进程写入字符串的次序，解释原因。

2、（1）参照参考程序2，编写程序。父进程从外界获取字符串，并将其写入共享文件；子进程从共享文件中获取字符串，并将其打印出来。

（2）修改程序，去掉所有lock/unlock调用。观察分析运行结果，解释原因。

（3）修改程序，不是去掉lock/unlock调用，而是去掉子进程中的”sleep(1);”。观察分析运行结果，解释原因。

## 参考程序

参考程序1

#include<fcntl.h>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int fatal (const char\* info)

{

perror (info);

exit (1);

}

int lock(int fd)

{

lseek(fd，0，SEEK\_SET);

if(lockf(fd，F\_LOCK，0)==-1)

fatal("lockf()");

return 0;

}

int unlock(int fd)

{

lseek(fd，0，SEEK\_SET);

if(lockf(fd，F\_ULOCK，0)==-1)

fatal("unlockf()");

return 0;

}

int main()

{

int fd;

int p1，p2，i;

char str[20];

if((fd=open("locked\_file.txt"，O\_RDWR|O\_APPEND|O\_CREAT，0666))<0)

fatal("open");

write(fd，"=========\n"，10);

while((p1=fork( ))== -1); /\*创建子进程p1\*/

if (p1==0)

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str，"daughter %d\n"，i);

write(fd，str，strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

}

else

{

while((p2=fork( ))==-1); /\*创建子进程p2\*/

if (p2==0)

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str，"son %d\n"，i);

write(fd，str，strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

}

else

{

lock(fd); /\*加锁\*/

for(i=0;i<3;i++)

{

sprintf(str，"parent %d\n"，i);

write(fd，str，strlen(str));

sleep(1);

}

unlock(fd); /\*解锁\*/

wait(NULL);

wait(NULL);

}

}

close(fd);

}

参考程序2

#include<fcntl.h>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int fatal (const char\* info)

{

perror (info);

exit (1);

}

int lock(int fd)

{

lseek(fd，0，SEEK\_SET);

if(lockf(fd，F\_LOCK，0)==-1)

fatal("lockf()");

return 0;

}

int unlock(int fd)

{

lseek(fd，0，SEEK\_SET);

if(lockf(fd，F\_ULOCK，0)==-1)

fatal("unlockf()");

return 0;

}

int main( )

{

int pid，fd;

char str[80] = {0};

fd=open("tmp.txt"，O\_RDWR|O\_CREAT|O\_TRUNC，0644);

pid=fork();

switch(pid)

{

case -1:

fatal("fork fail!");

case 0:

sleep(1);

lock(fd);

lseek(fd， SEEK\_SET，0);

read(fd，str，sizeof(str));

unlock(fd);

printf("son %d:read str from tmpfile:%s\n"，getpid()，str);

exit(0);

default:

lock(fd);

printf("parent %d :please enter a str for tmpfile(strlen<80):\n"，getpid());

//~~scanf("%s"，str);~~

Fgets(str， sizeof(str)-1， stdin);

lseek(fd， 0， SEEK\_SET);

write(fd，str，strlen(str));

unlock(fd);

wait(0);

close(fd);

exit(0);

}

}

## 思考题（实验报告）

1、函数unlock和lock的实现中，lseek的作用？

2、解释参考程序1，2中lock/unlock的作用？

3、参考程序2，子进程中的”sleep(1);”的作用？