【写者优先】在读者、写者问题中，如果总有读者进程进行读操作，会造成写者进程永远都不能进行写操作（读者优先），即所谓的写者饿死现象。给出读者、写者问题的另一个解决方案：即保证当有一个写者进程想写时，不允许读者进程再进入，直到写者写完为止，即写者优先。

让我们先回顾读者写者问题[1]：

一个数据对象若被多个并发进程所共享，且其中一些进程只要求读该数据对象的内容，而另一些进程则要求写操作，对此，我们把只想读的进程称为“读者”，而把要求写的进程称为“写者”。在读者、写者问题中，任何时刻要求“写者”最多只允许有一个执行，而“读者”则允许有多个同时执行。因为多个“读者”的行为互不干扰，他们只是读数据，而不会改变数据对象的内容，而“写者”则不同，他们要改变数据对象的内容，如果他们同时操作，则数据对象的内容将会变得不可知。所以对共享资源的读写操作的限制条件是：

* + 允许任意多的读进程同时读；
  + 一次只允许一个写进程进行写操作；
  + 如果有一个写进程正在进行写操作，禁止任何读进程进行读操作。

为了解决该问题，我们只需解决“写者与写者”和“写者与第一个读者”的互斥问题即

可，为此我们引入一个互斥信号量Wmutex，为了记录谁是第一个读者，我们用一个共享整

型变量Rcount 作一个计数器。而在解决问题的过程中，由于我们使用了共享变量Rcount，

该变量又是一个临界资源，对于它的访问仍需要互斥进行，所以需要一个互斥信号量Rmutex，

算法如下：

semaphore Wmutex, Rmutex = 1;

int Rcount = 0;

void reader() /\*读者进程\*/

{

while (true)

{

P(Rmutex);

if (Rcount == 0) P(wmutex);

Rcount = Rcount + 1;

V(Rmutex);

⋯⋯;

read; /\* 执行读操作 \*/

⋯⋯;

P(Rmutex);

Rcount = Rcount - 1;

if (Rcount == 0) V(wmutex);

V(Rmutex);

}

}

void writer() /\*写者进程\*/

{

while (true)

{

P(Wmutex);

⋯⋯;

write; /\* 执行写操作 \*/

⋯⋯;

P(Wmutex);

}

}

现在回到【写者优先】优先问题

【写者优先】在读者、写者问题中，**如果总有读者进程进行读操作，会造成写者进程永远都不能进行写操作（读者优先），即所谓的写者饿死现象**。给出读者、写者问题的另一个解决方案：即保证当有一个写者进程想写时，不允许读者进程再进入，直到写者写完为止，即写者优先。

【解题思路】在上面的读者写者问题基础上，做以下修改：

* + 增加授权标志authFlag，当写者到来，发现有读者在读，则取消授权，然后等待缓冲区；
  + 增加“等待授权计数器waitAuthCount”，写者离开时，如果waitAuthCount大于0，则迭代唤醒等待授权的读者；
  + 读者到来，首先看授权标志，如果有授权标志，则继续，否则等待授权，即写者取消授权后，新来的读者不能申请缓冲区。
  + 增加Amutex互斥信号量，保护waitAuthCount、authFlag
  + 增加同步信号量mutexAuth，表示等待授权的读者队列

【伪代码如下】

semaphore Wmutex=1, Rmutex=1 ,Amutex=1, mutexAuth=0;

int Rcount = 0;

int waitAuthCount=0;

int authFlag=1；

void reader() /\*读者进程\*/

{

P(Amutex)//临界资源authFlag和waitAuthCount互斥

if (authFlag==0)

{

waitAuthCount++;

p(mutexAuth);

}

V(Amutex)

P(Rmutex);//临界资源Rcount互斥

if (Rcount == 0) P(wmutex);//缓冲区互斥

Rcount = Rcount + 1;

V(Rmutex);

......;

read; /\* 执行读操作 \*/

......;

P(Rmutex);

Rcount = Rcount - 1;

if (Rcount == 0)

V(wmutex);

V(Rmutex);

}

}

void writer() /\*写者进程\*/

{

if(Rcount>0) //有读者正在读

authFlag=0；

P(Wmutex);

......;

write; /\* 执行写操作 \*/

......;

P(Wmutex);

P(Amutex)

while(waitAuthCount>0){ //写完后，发现有读者在等

v(mutexAuth);

waitAuthCount--;

}

V(Amutex)

}

【vc++代码】

// testsemaphore.cpp : Defines the entry point for the console application.

// by 西南大学计算机科学系周竹荣

//

#include "stdafx.h"

#include "windows.h"

#include "process.h"

#include<iostream>

#include <time.h>

using namespace std;

#define P(S) WaitForSingleObject(S,INFINITE)//定义Windows下的P操作

#define V(S) ReleaseSemaphore(S,1,NULL)//定义Windows下的V操作

HANDLE Wmutex, Rmutex ,Authmutex,Amutex;

int Rcount = 0;

int waitAuthCount=0;

int authFlag=1;

DWORD WINAPI reader(){

P(Amutex);

if (authFlag==0)

{

waitAuthCount++;

cout<<"等待授权,线程ID："<<GetCurrentThreadId()<<endl;

P(Authmutex); //挂起

}

V(Amutex);

P(Rmutex);

if (Rcount == 0)

P(Wmutex);

Rcount = Rcount + 1;

V(Rmutex);

cout<<"读数据,线程ID："<<GetCurrentThreadId()<<endl;

Sleep(500); /\* 执行读操作 \*/

P(Rmutex);

Rcount = Rcount - 1;

if (Rcount == 0)

V(Wmutex);

V(Rmutex);

return 1;

};

DWORD WINAPI writer(){

if(Rcount>0)

{

cout<<"取消授权,线程ID："<<GetCurrentThreadId()<<endl;

authFlag=0;

}

P(Wmutex);

cout<<"写数据,线程ID："<<GetCurrentThreadId()<<endl;

Sleep(2000); /\* 执行写操作 \*/

V(Wmutex);

cout<<"授权读,线程ID："<<GetCurrentThreadId()<<endl;

P(Amutex);

authFlag=1;

while(waitAuthCount>0){

V(Authmutex);

waitAuthCount--;

}

V(Amutex);

return 1;

};

int main(int argc, char\* argv[])

{

Wmutex=CreateSemaphore(NULL,1,1,NULL);

Rmutex=CreateSemaphore(NULL,1,1,NULL);

Authmutex=CreateSemaphore(NULL,1,1,NULL);

Amutex=CreateSemaphore(NULL,1,1,NULL);

while(1){

Sleep(100);

srand((unsigned)time(NULL));

int rC=rand()%1000;

Sleep(rC);

if( rC % 6==0)

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)writer,NULL,NULL,NULL);

else

CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)reader,NULL,NULL,NULL);

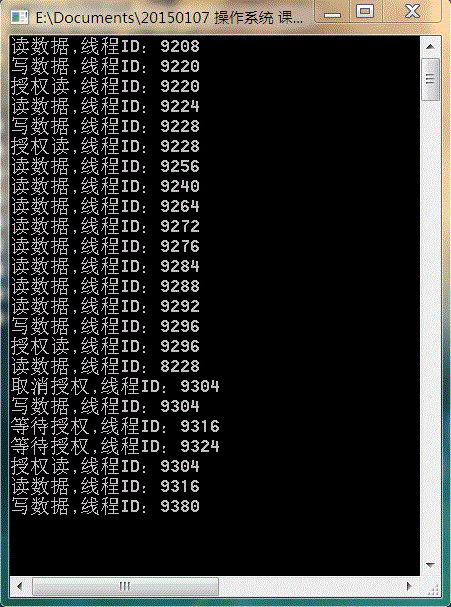
}

Sleep(600000);

return 0;

}

【运行结果】



参考文献

[1]计算机操作系统-郁红英-冯庚豹-人民邮电出版社