**操作系统课程设计报告** 日期：19.4.25

|  |  |  |  |
| --- | --- | --- | --- |
| 实习题目：读者写者问题 | | | |
| 完成人 | 姓名：龙佳琪 | 班级：173040107 | 学号2017141461065 |
| 实习内容简要描述 | 创建一个包含n个线程的控制台进程，这n个线程表示读者进程或写者进程，通过信号量机制分别实现读者优先和写者优先的 读者写者问题。  其中具体要求为：  写-写互斥，即不能有两个写者同时进行写操作  读-写互斥，即不能同时有一个线程在读，而另一个线程在写  读-读互斥，即可以有一个或多个读者在读  更具体的条件为：  读者优先：   1. 当读者数为0时可以先进行写进程，直到第一个读者到来，在当前进行的写者结束后立即执行 2. 当所有的读者进程执行完后可以开始执行写者进程，按照开始进行等待的序列进行   写者优先：  同理可得 | | |
| 主要代码结构  （附注释） | **全局变量：**    用于计算读、写进程的总数    RP\_Write是以临界区的形式来互斥的，由于我想试验这两种方式，所以读者优先中选用临界区互斥  **功能函数：**    分别为读者优先中的读者进程、读者优先中的写者进程、当用户选择读者优先    分别为写者优先中的读者进程、写者优先中的写者进程、当用户选择写者优先  **线程结构体** | | |
| 结果分析（或错误原因分析） | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 。。。 | | 1读re | 2写re | 3读re | 4读re | 3读完 | 1读完 |  |  | 4读完 |  |  |  |  | 2写完 | 5写完 | | 1开读 |  | 3开读 | 4开读 |  |  |  |  | 2开始写 |  |  |  |  | 5开始写 |  | |  |  | 5写re |  |  |  |  |  |  |  |  |  |  |  |  |      |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 3 | 1读re | 1开读 |  |  |  |  |  |  | | 4 | 2写re |  |  |  |  |  |  |  | | 5 | 3读re | 5写re |  |  |  |  |  |  | | 6 | 4读re |  |  |  |  |  |  |  | | 7 |  |  |  |  |  |  |  |  | | 8 | 1读完 | 2开写 |  |  |  |  |  |  | | 9 |  |  |  |  |  |  |  |  | | 10 |  |  |  |  |  |  |  |  | | 11 |  |  |  |  |  |  |  |  | | 12 | 2写完 | 5开写 |  |  |  |  |  |  | | 。。。 |  |  |  |  |  |  |  |  | | 15 | 5写完 | 3开读 | 4开读 |  |  |  |  |  | | 。。。 | 3读完 |  |  |  |  |  |  |  | | 。。。 | 4读完 |  |  |  |  |  |  |  | | | |
| 注释：如果上述表格空间不够，可以另附表格进行说明 | | | |

设计文档：

从Pascal伪代码的方式进行分析

通过资料的查找可知，进行信号量互斥设置的方法有两种，一种是通过临界区CRITICAL\_SECTION；或者是通过HANDLE

1. CRITICAL\_SECTION cs;

进入临界区：

EnterCriticalSection(\*CRITICAL\_SECTION);

退出临界区：

LeaveCriticalSection(\*CRITICAL\_SECTION);

1. HANDLE mutex；

CreatMutex()

OpenMutex()

WaitForSingleObject()

RealseMutex()

读者优先：

读者进程：

（rmutex是互斥readcount的，wmutex是控制读写互斥）

Reader:

P(rmutex)

If readcount =0 then P(wmutex);

Readcount=readcount+1;

V(rmutex)

Read text

P(rmutex)

Readcount=readcount-1;

If readcount =0 then V(wmutex);

V(rmutex)

Writer:

P(wmutex)

Write text

V(wmutex)

写者优先：

（rmutex——readcount, wmutex——writecount, RWmutex——读写互斥，mutex写写互斥，mutex2——避免多个读者的竞争）

Reader:

P(mutex2)

P(RWmutex)

P(rmutex)

Readcount=readcount+1

If readcount =1 then P(mutex)

V(rmutex)

V(RWmutex)

V(mutex2)

Read text

P(rmutex)

Readcount=readcount-1;

If readcount=0 then P(mutex)

V(rmutex)

Writer:

P(wmutex)

Writecount=writecount+1

If writecount=1 then P(RWmutex)

V(wmutex)

P(mutex)

Write text

V(mutex)

P(wmutex)

Writecount=writecount-1

If writecount=0 then V(RWmutex)

V(wmutex)

附代码:

#include<cstdio>

#include<Windows.h>

#include<ctime>

#include<fstream>

#include<conio.h>

#include<io.h>

using namespace std;

int readcount = 0, writecount = 0;

//第一种实现互斥的方式，临界区

CRITICAL\_SECTION RP\_Write;

void RP\_readThread(void \*p);

void RP\_writeThread(void \*p);

void ReaderPriority(char \*flie);

void WP\_readThread(void \*p);

void WP\_writeThread(void \*p);

void WriterPriority(char \*file);

struct ThreadInfo

{

int order; //线程序号

char type; //R类型还是W类型

double begin; //开始的时间，也就是一开始要先等待这段时间

double persist; //持续的时间

};

int main()

{

while (true)

{

char ch;

printf("readPriorty-----1-------------------------------\nwritePriorty-----2---------------------------------------------\nexit-----------------------3-------\n");

ch = \_getch();

if (ch == '1')

{

ReaderPriority((char \*)"thread.txt");

}

else if (ch == '2')

{

WriterPriority((char \*)"thread.txt");

}

else if (ch == '3')

{

break;

}

else

{

printf("error\n\n");

}

}

system("pause");

return 0;

}

void RP\_readThread(void \*p)

{

HANDLE rmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_readcount");//readcount是一个临界变量，修改的时候要用互斥量rmutex

int order = ((ThreadInfo\*)p)->order;

DWORD begin = (DWORD)((((ThreadInfo\*)p)->begin)\*1000);

DWORD persist = (DWORD)((((ThreadInfo\*)p)->persist)\*1000);

Sleep(begin);

printf("Reader thread %d sents the reading require.\n", order);

DWORD wait\_for\_rmutex = WaitForSingleObject(rmutex, -1);//要等到rmutex才可以进行readcount的修改

readcount++;

if (readcount == 1)//第一个读者

{

EnterCriticalSection(&RP\_Write);//写锁上，也就是代表使用了这个临界区，完成读写互斥

}

ReleaseMutex(rmutex); //释放readcount的互斥信号

//读者读书

printf("Reader thread %d begins to read the file.\n", order);

Sleep(persist);

//读完了退出

printf("Reader thread %d finished read file.\n", order);

//等待互斥量来修改readcount

wait\_for\_rmutex = WaitForSingleObject(rmutex, -1);

readcount--;

if (readcount == 0)//读者全部读完，唤醒写者

{

LeaveCriticalSection(&RP\_Write);

}

ReleaseMutex(rmutex);

}

void RP\_writeThread(void \*p)

{

int order = ((ThreadInfo\*)(p))->order;

DWORD begin = (DWORD)((((ThreadInfo\*)(p))->begin) \* 1000);

DWORD persist = (DWORD)((((ThreadInfo\*)(p))->persist) \* 1000);

Sleep(begin);

//等待资源

printf("Writer thread %d sents the writing require.\n", order);

EnterCriticalSection(&RP\_Write);

//写文件

printf("Write thread %d begins to Write to th file.\n", order);

Sleep(persist);

printf("Write thread %d finished Writing to the file.\n", order);

LeaveCriticalSection(&RP\_Write);

}

void ReaderPriority(char \*file)

{

DWORD n\_thread = 0;//线程数

DWORD thread\_ID;

DWORD wait\_for\_all;

HANDLE rmutex = CreateMutex(NULL, false, "mutex\_for\_readcount");

//线程对象数组

HANDLE h\_Thread[100];

ThreadInfo thread\_info[100];

readcount = 0;

//临界区的初始化

InitializeCriticalSection(&RP\_Write);

ifstream inFile;

inFile.open(file);

printf("Reader Priority:\n\n");

while (inFile)

{

inFile >> thread\_info[n\_thread].order;

inFile >> thread\_info[n\_thread].type;

inFile >> thread\_info[n\_thread].begin;

inFile >> thread\_info[n\_thread].persist;

n\_thread++;

inFile.get();

}

for (int i = 0; i < (int)n\_thread; i++)

{

if (thread\_info[i].type == 'R')

{

h\_Thread[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)(RP\_readThread), &thread\_info[i], 0, &thread\_ID);

}

else

{

h\_Thread[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)(RP\_writeThread), &thread\_info[i], 0, &thread\_ID);

}

}

wait\_for\_all = WaitForMultipleObjects(n\_thread, h\_Thread, TRUE, -1);

printf("finish all!\n");

}

void WP\_readThread(void \*p)

{

//全部通过HANDLE也可以实现互斥

HANDLE wmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_writecount");

HANDLE RWmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_read\_and\_write");

HANDLE mutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_write\_and\_write");

HANDLE rmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_readcount");

HANDLE mutex2 = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_reads");

int order = ((ThreadInfo\*)p)->order;

DWORD begin = (DWORD)((((ThreadInfo\*)p)->begin) \* 1000);

DWORD persist = (DWORD)((((ThreadInfo\*)p)->persist) \* 1000);

Sleep(begin);

printf("Reader thread %d sents the reading require.\n", order);

DWORD wait\_for\_mutex2 = WaitForSingleObject(mutex2, -1);

DWORD wait\_for\_RWmutex = WaitForSingleObject(RWmutex, -1);//读写互斥

DWORD wait\_for\_rmutex = WaitForSingleObject(rmutex, -1);//readcount互斥

DWORD wait\_for\_mutex;//写写互斥

readcount++;

if (readcount == 1) //如果还没有写者，那么读者先读，同时阻塞写

{

wait\_for\_mutex = WaitForSingleObject(mutex, INFINITE);//对mutex进行占用，也就阻塞，与前锁临界区的效果相似

}

ReleaseMutex(rmutex); //释放readcount的互斥信号

ReleaseMutex(RWmutex);

ReleaseMutex(mutex2);

printf("Reader thread %d begins to read the file.\n", order);

Sleep(persist);

//读完了退出

printf("Reader thread %d finished read file.\n", order);

wait\_for\_mutex = WaitForSingleObject(rmutex, -1);//等待互斥量来修改readcount

readcount--;

if(readcount==0)

ReleaseMutex(mutex);

ReleaseMutex(rmutex);

}

void WP\_writeThread(void \*p)

{

HANDLE rmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_readcount");

HANDLE wmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_writecount");

HANDLE RWmutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_read\_and\_write");

HANDLE mutex = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_write\_and\_write");

HANDLE mutex2 = OpenMutex(MUTEX\_ALL\_ACCESS, false, "mutex\_for\_reads");

int order = ((ThreadInfo\*)p)->order;

DWORD begin = (DWORD)((((ThreadInfo\*)p)->begin) \* 1000);

DWORD persist = (DWORD)((((ThreadInfo\*)p)->persist) \* 1000);

Sleep(begin);

printf("Writer thread %d sents the writing require.\n", order);

DWORD wait\_for\_wmutex = WaitForSingleObject(wmutex, -1);//用来修改writecount

writecount++;

if (writecount == 1)

{

DWORD wait\_for\_RWmutex = WaitForSingleObject(RWmutex, INFINITE); //读写互斥

}

ReleaseMutex(wmutex);

DWORD wait\_for\_mutex = WaitForSingleObject(mutex, -1);//写写互斥

printf("Writer thread %d begins to write the file.\n", order);

Sleep(persist);

printf("Writer thread %d finished writing file.\n", order);

ReleaseMutex(mutex);

wait\_for\_mutex = WaitForSingleObject(wmutex, -1);

writecount--;

if (writecount == 0) //写者全部写完了就可以释放读的互斥信号，读进程可以开始

{

ReleaseMutex(RWmutex);

}

ReleaseMutex(wmutex);

}

void WriterPriority(char \*file)

{

DWORD n\_thread = 0;//线程数

DWORD thread\_ID;

DWORD wait\_for\_all;

HANDLE rmutex = CreateMutex(NULL, false, "mutex\_for\_readcount");

HANDLE wmutex = CreateMutex(NULL, false, "mutex\_for writecount");

HANDLE mutex2 = CreateMutex(NULL, false, "mutex\_for\_reads");//防多竞争

HANDLE RWmutex = CreateMutex(NULL, false, "mutex\_for\_read\_and\_write");//读写互斥

HANDLE mutex = CreateMutex(NULL, false, "mutex\_for\_write\_and\_write");//写写互斥

//线程对象数组

HANDLE h\_Thread[100];

ThreadInfo thread\_info[100];

readcount = 0;

writecount = 0;

ifstream inFile;

inFile.open(file);

printf("Writer Priority:\n\n");

while (inFile)

{

inFile >> thread\_info[n\_thread].order;

inFile >> thread\_info[n\_thread].type;

inFile >> thread\_info[n\_thread].begin;

inFile >> thread\_info[n\_thread].persist;

n\_thread++;

inFile.get();

}

for (int i = 0; i < (int)n\_thread; i++)

{

if (thread\_info[i].type == 'R')

{

h\_Thread[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)(WP\_readThread), &thread\_info[i], 0, &thread\_ID);

}

else

{

h\_Thread[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)(WP\_writeThread), &thread\_info[i], 0, &thread\_ID);

}

}

wait\_for\_all = WaitForMultipleObjects(n\_thread, h\_Thread, TRUE, -1);

printf("WP finish all!\n");

}

运行结果截屏：

读者优先：



写者优先：

 