读者写者-进程同步


```
班级: 2015211306
学号: 2015211301
姓名: 魏　晓
时间: 2018.01.10
```


- **实验亮点**
  - 重新用C++实现读者-写者
  - 重构核心代码段,具体见下文

实验要求

在 Windows 环境下，创建一个包含 n 个线程的控制进程。用这 n 个线程来表示 n 个读者或写者。每个线程按相应测试数据文件的要求，进行读写操作。请用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。

读者-写者问题的读写操作限制：
- 写-写互斥；
- 读-写互斥；
- 读-读允许；

读者优先的附加限制：如果一个读者申请进行读操作时已有另一读者正在进行读操作，则该读者可直接开始读操作。

写者优先的附加限制：如果一个读者申请进行读操作时已有另一写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。

运行结果显示要求：要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息，以确信所有处理都遵守相应的读写操作限制。

测试数据文件格式

测试数据文件包括n 行测试数据，分别描述创建的n 个线程是读者还是写者，以及读写操作的开始时间和持续时间。每行测试数据包括四个字段，各字段间用空格分隔。

- 第一字段为一个正整数，表示线程序号。第一字段表示相应线程角色，R 表示读者是，W 表示写者。
- 第二字段为一个正数，表示读写操作的开始时间。线程创建后，延时相应时间（单位为秒）后发出对共享资源的读写申请。
- 第三字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。

下面是一个测试数据文件的例子：

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 5.1 3
```

实验相关的API

- 在本实验中可能涉及的API 有：
    - 线程控制：CreateThread 完成线程创建，在调用进程的地址空间上创建一个线程，以执行指定的函数；它的返回值为所创建线程的句柄。

      ```
      HANDLE CreateThread (
      LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
      DWORD dwStackSize, // initial stack size
      LPTHREAD_START_ROUTINE lpStartAddress, // thread
      function
      LPVOID lpParameter, // thread argument
      DWORD dwCreationFlags, // creation option
      LPDWORD lpThreadId // thread identifier
      ) ;
      ```

    - ExitThread 用于结束当前线程。

      ```
      VOID ExitThread (
      DWORD dwExitCode // exit code for this thread
      ) ;
      ```

- Sleep 可在指定的时间内挂起当前线程。

```
VOID Sleep (
DWORD dwMilliseconds // sleep time
) ;
```

- 信号量控制：
  - CreateMutex 创建一个互斥对象，返回对象句柄；

```
HANDLE CreateMutex (
LPSECURITY_ATTRIBUTES lpMutexAttributes, // SD
BOOL bInitialOwner, // initial owner
LPCTSTR lpName // object name
) ;
```

- OpenMutex 打开并返回一个已存在的互斥对象句柄，用于后续访问；

```
HANDLE OpenMutex (
DWORD dwDesiredAccess, // access
BOOL bInheritHandle, // inheritance option
LPCTSTR lpName // object name
) ;
```

- ReleaseMutex 释放对互斥对象的占用，使之成为可用。

```
BOOL ReleaseMutex (
HANDLE hMutex // handle to mutex
) ;
```

- WaitForSingleObject 可在指定的时间内等待指定对象为可用状态；

```
DWORD WaitForSingleObject (
HANDLE hHandle, // handle to object
DWORD dwMilliseconds // time-out interval
) ;
```

程序

在老师给出参考示例的基础上,我使用 C++ **重新构建了一遍代码**.并且替换了部分库

`Reader-Writer.cpp`

算法

读者优先

- 原先的代码
  - ReaderPriority函数首先读取目标文件Thread.dat，为每一行请求创建一个线程，其中读请求创建读者线程，调用RP_ReaderThread函数，写请求创建写者线程，调用RP_WriterThread函数。
    - RP_ReaderThread函数的实现如下：

```
        P(mutex);
        read_count++;
        if(read_count==1)
           P(&RP_Write);
        V(mutex);
        读临界区……
        P(mutex);
        read_count--;
        if(read_count==0)
           V(&RP_Write);
        V(mutex);
```

- RP_WriterThread函数的实现如下：

```c
    P(&RP_Write);
    写临界区……
    V(&RP_Write);

```

- 改进后的代码

```
// Resource to R/W
auto resource = CreateSemaphore (NULL, 1, 1, NULL);

// Reader Count
auto rmutex = CreateSemaphore (NULL, 1, 1, NULL);
size_t read_count = 0;
```

读者

```
WaitForSingleObject (rmutex, INFINITE);
read_count++;
if (read_count == 1)
    WaitForSingleObject (resource, INFINITE);
ReleaseSemaphore (rmutex, 1, NULL);

// Critical Section

WaitForSingleObject (rmutex, INFINITE);
read_count--;
if (read_count == 0)
    ReleaseSemaphore (resource, 1, NULL);
ReleaseSemaphore (rmutex, 1, NULL);
```

写者

```
WaitForSingleObject (resource, INFINITE);

// Critical Section

ReleaseSemaphore (resource, 1, NULL);
```

写者优先

- 题目
    - WriterPriority函数首先读取目标文件Thread.dat，为每一行请求创建一个线程，其中读请求创建读者线程，调用WP_ReaderThread函数，写请求创建写者线程，调用WP_WriterThread函数。

- WP_ReaderThread函数实现如下：

```
    P(mutex1);
    P(&cs_Read);
    P(mutex2);
    read_count++;
    if(read_count==1)
      P(&cs_Write);
    V(mutex2);
    V(&cs_Read);
    V(mutex1);
    读临界区……
    P(mutex2);
    read_count--;
    if(read_count==0)
        V(&cs_Write);
      V(mutex2);
```

- WP_WriterThread函数实现如下：

```
P(mutex3);
write_count++;
if(write_count==1)
  P(&cs_Read);
V(mutex3);
P(&cs_Write);
写临界区……
```

- 改进后的代码

```
// Resource to R/W
auto resource = CreateSemaphore (NULL, 1, 1, NULL);

// Reader trying to enter
auto readTry = CreateSemaphore (NULL, 1, 1, NULL);

// Reader Count
auto rmutex = CreateSemaphore (NULL, 1, 1, NULL);
size_t read_count = 0;

// Writer Count
auto wmutex = CreateSemaphore (NULL, 1, 1, NULL);
size_t write_count = 0;
```

读者

```
WaitForSingleObject (readTry, INFINITE);

WaitForSingleObject (rmutex, INFINITE);
read_count++;
if (read_count == 1)
    WaitForSingleObject (resource, INFINITE);
ReleaseSemaphore (rmutex, 1, NULL);

ReleaseSemaphore (readTry, 1, NULL);

// Critical Section

WaitForSingleObject (rmutex, INFINITE);
read_count--;
if (read_count == 0)
    ReleaseSemaphore (resource, 1, NULL);
ReleaseSemaphore (rmutex, 1, NULL);
```

写者

```
WaitForSingleObject (wmutex, INFINITE);
write_count++;
if (write_count == 1)
    WaitForSingleObject (readTry, INFINITE);
ReleaseSemaphore (wmutex, 1, NULL);

WaitForSingleObject (resource, INFINITE);

// Critical Section

ReleaseSemaphore (resource, 1, NULL);

WaitForSingleObject (wmutex, INFINITE);
write_count--;
if (write_count == 0)
    ReleaseSemaphore (readTry, 1, NULL);
ReleaseSemaphore (wmutex, 1, NULL);
```

输入输出

输入

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 5.1 3
```

输出

```
Reader Preference
Thread 1 - Reader  - Request
Thread 1 - Reader  - Begin
Thread 2 - Writer  - Request
Thread 3 - Reader  - Request
Thread 3 - Reader  - Begin
Thread 5 - Writer  - Request
Thread 4 - Reader  - Request
Thread 4 - Reader  - Begin
Thread 3 - Reader  - End
Thread 1 - Reader  - End
Thread 4 - Reader  - End
Thread 2 - Writer  - Begin
Thread 2 - Writer  - End
Thread 5 - Writer  - Begin
Thread 5 - Writer  - End
Writer Preference
Thread 1 - Reader  - Request
Thread 1 - Reader  - Begin
Thread 2 - Writer  - Request
Thread 3 - Reader  - Request
Thread 5 - Writer  - Request
Thread 4 - Reader  - Request
Thread 1 - Reader  - End
Thread 2 - Writer  - Begin
Thread 2 - Writer  - End
Thread 5 - Writer  - Begin
Thread 5 - Writer  - End
```

```
Thread 3 - Reader  - Begin
Thread 4 - Reader  - Begin
Thread 3 - Reader  - End
Thread 4 - Reader  - End
End
```

实验感悟

- 这次操作系统实验的首要任务就是按照提示文档读懂示例代码,其中用到的是很久没接触的c语言知识
- 在运行实例代码的时候,有两个错误,不适应我的机器,在调整之后才可以正确运行
- 在实际编写程序的时候要仔细理解读者写者不同的优先级
- 最后调整运行下来,感觉**读懂源代码是关键**
- 让我对这学期所学的操作系统知识结合到现实编程中,真正的学有所思,学有所用