# 天津商业大学信息工程学院专业实验室学生实验报告

实验日期：2014 年 12 月 3 日　　实验时间：13:30——15:05
课程名称：操作系统　　任课教师：苗序娟　　实验成绩：
专业班级：软件工程 1201 班　　学生姓名：王靖伟

**实验项目名称：**实验三　操作系统中的经典线程同步问题

**实验设备：**PC 一台，Windows 2000/XP 及以上操作系统, VC++ 6.0

**实验目的：**

1、加深对线程的理解、掌握 Windows 中线程的操作。

2、掌握死锁产生的原因。

3、掌握信号量、互斥量、事件、临界区等同步对象的使用。

**实验要求：**

1、进程和线程的关系。

2、线程间的同步和通信。

3、本实验内容主要对应于教材第 4、5、6、7 章

**实验描述：**

1、运行实验程序“运行.exe”，出现如下界面，如图 1 所示：


图 1. 运行实验程序“运行.exe”

2、点击运行读者-写者程序按钮，出现如下界面，如图 2 所示：

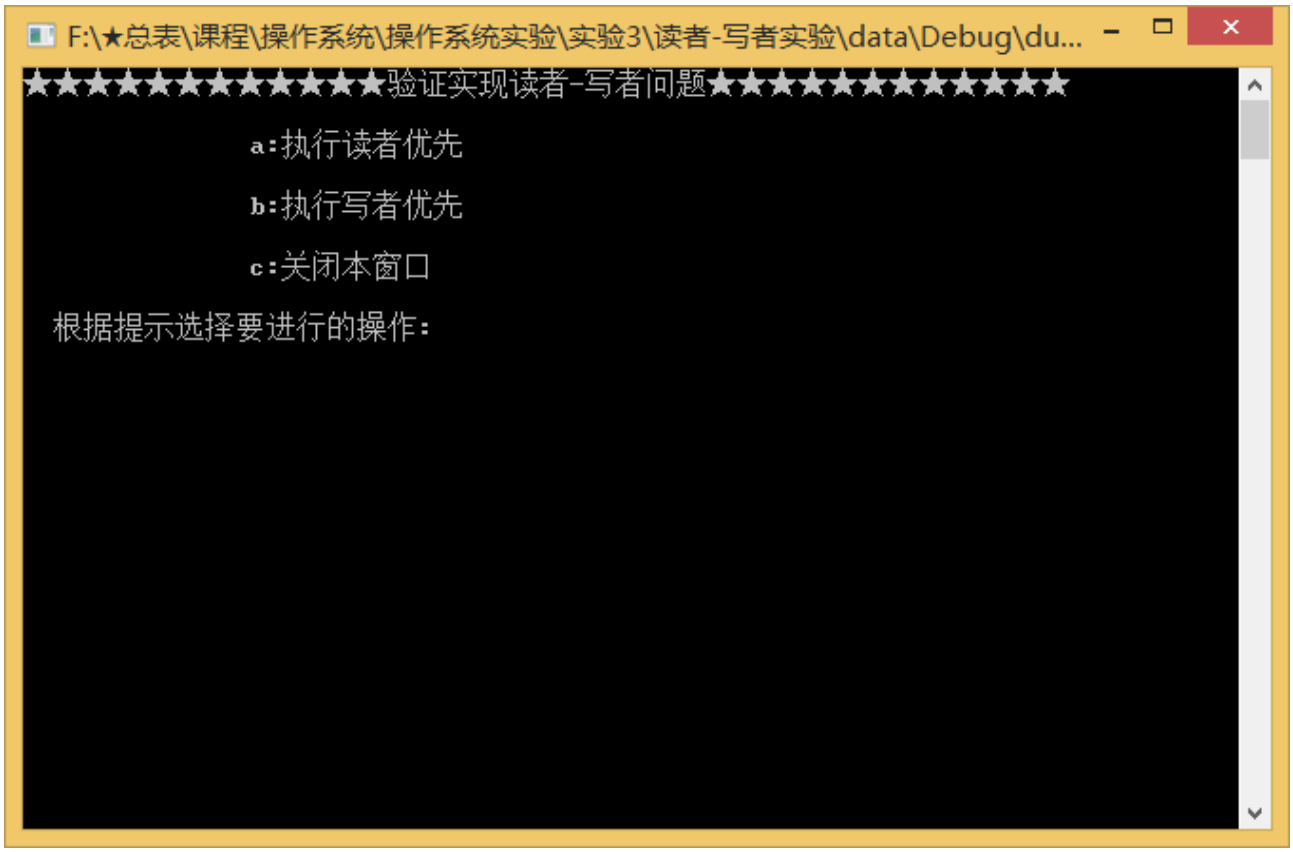


图 2. 点击“运行读者-写者程序”按钮

选择 a 或 b 操作，进行实验，观察结果的输出。

3、观察第二步的实验现象，多次试验，可总结为：

(1)、选择 a 操作和 b 操作的输出结果的区别是 多次试验后，发现 a 操作的结果都相同，b 操作的结果也都相同，于是为了更好的观察 a 操作和 b 操作的输出结果的区别，对 a 操作和 b 操作同时截图，如图 3 所示。发现当进行 a 操作时，读者优先读取数据，当读者完成所有操作之后，才允许写者进行写操作。当进行 b 操作时，写者优先写入数据，但由于初始时刻读者发出都请求，则读者 1 先进行开始读操作，在读者 1 读取完成之前，写者 2、读者 3、读者 4、写者 5 发出读/写请求后，优先完成写者请求，当写者的写操作全部完成后再进行读者的读操作 。

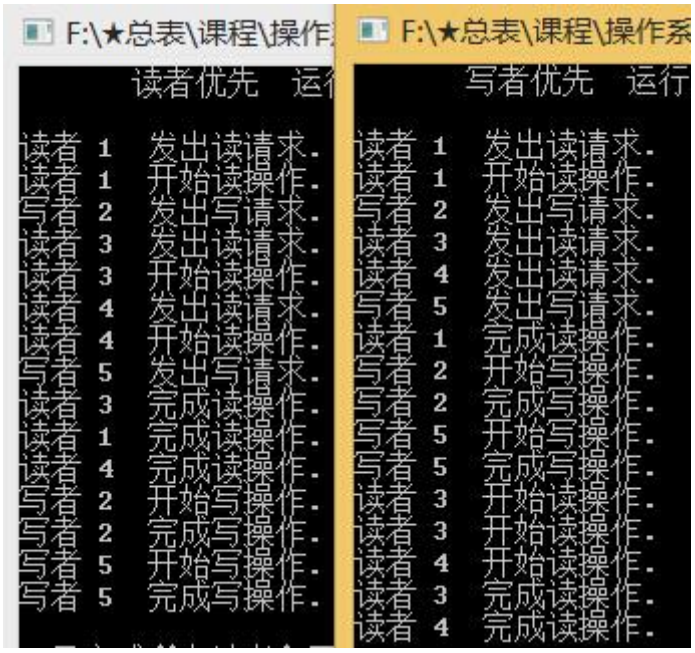


图 3. a 操作和 b 操作的结果

(2)、修改目录“data”中的“yanzheng.txt”文档中的数据，观察执行 a 操作和 b 操作之后有什么变化 改变第一列数据发现运行后的线程序号改变了，改变第二列数据发现运行后的线程类别（读者/写者）改变，改变第三列数据发现运行后的线程开始时间有变化了，改变第四列数据发现运行后的线程读/写持续时间有变化了 ，说明“yanzheng.txt”文档中各列数据的含义是什么。

duzhe-xiezhe.cpp 中分析，13—18 行定义结构体如代码清单 1 所示，90—93 行读取文档如代码清单 2 所示。

通过修改“yanzheng.txt”文档中各列数据实验和通过查看代码都很容易发现，“yanzheng.txt”文档中第一列数据是**判断线程是读者还是写者（即线程类别）**，第二列数据是**线程序号**，第三列数据是**线程开始时间**，随后一列数据是**线程读/写持续时间**。

代码清单 1

| 行数 | duzhe-xiezhe.cpp 中 13—18 行 |
|---|---|
| 13 | `struct ProcessInfo //线程信息` |
| 14 | `{ int Process_serial_number; //线程序号` |
| 15 | `double Process_Delay; //线程读/写持续时间` |
| 16 | `double Process_Start; //线程开始时间` |
| 17 | `char ProcessClass; //判断线程是读者还是写者（即线程类别）` |
| 18 | `};` |

代码清单 2

| 行数 | duzhe-xiezhe.cpp 中 90—93 行 |
|---|---|
| 90 | inFile>>process_info[Processnumber].Process_serial_number |
| 91 | >>process_info[Processnumber].ProcessClass |
| 92 | >>process_info[Processnumber].Process_Start |
| 93 | >>process_info[Processnumber].Process_Delay; |

(3)、这说明“读者”间的关系是 相容 、“写者”之间的关系是 相容 、“读者－写者”之间的关系是 互斥 。（填相容、互斥）

4、点击实验界面上的“运行 duozhe-xiezhe.cpp”按钮，查找并说明下列函数的用法：

(1) CreateThread();

| 行数 | CreateThread()函数原型及用法 |
|---|---|
| 1 | HANDLE CreateThread(//创建线程，返回新建线程的句柄 |
| 2 | LPSECURITY_ATTRIBUTES lpThreadAttributes,//线程的安全属性 |
| 3 | SIZE_T dwStackSize,//线程堆栈大小 |
| 4 | LPTHREAD_START_ROUTINE lpStartAddress,//线程执行的起点，也就是线程函数的指针 |
| 5 | LPVOID lpParameter,//启动线程函数的参数 |
| 6 | DWORD dwCreationFlags,//线程创建标志 |
| 7 | LPDWORD lpThreadId//返回线程 ID |
| 8 | ); |
| 9 | //参考文献：范文庆，周彬彬，安靖. Windows API 开发详解：函数、接口、编程实例[M]. 2013 |
| 10 | 年 1 月北京第 3 次印刷. 北京:人民邮电出版社，2011:180. |

(2) CreateMutex();

| 行数 | CreateMutex()函数原型及用法 |
|---|---|
| 1 | HANDLE WINAPI CreateMutex(//创建互斥对象 |
| 2 | LPSECURITY_ATTRIBUTES lpMutexAttributes,//安全属性，一般设置为 NULL |
| 3 | BOOL bInitialOwner,//创建后是否被创建线程所“拥有” |
| 4 | LPCTSTR lpName//互斥对象名 |
| 5 | ); |
| 6 | //参考文献：范文庆，周彬彬，安靖. Windows API 开发详解：函数、接口、编程实例[M]. 2013 |
| 7 | 年 1 月北京第 3 次印刷. 北京:人民邮电出版社，2011:223-224. |

(3) ReleaseMutex();

| 行数 | ReleaseMutex()函数原型及用法 |
|---|---|
| 1 | BOOL WIANPI ReleaseMutex(//释放互斥对象 |
| 2 | HANDLE hMutex |
| 3 | ); |
| 4 | //一个线程释放了互斥对象后，如果其他进程在等待互斥对象位置，则等待的线程可以得到该互斥对 |
| 5 | 象，等待函数返回，互斥对象被新的线程所拥有。 |
| 6 | |
| 7 | //参考文献：范文庆，周彬彬，安靖. Windows API 开发详解：函数、接口、编程实例[M]. 2013 |
| 8 | 年 1 月北京第 3 次印刷. 北京:人民邮电出版社，2011:224. |

(4) WaitForSingleObject();

| 行数 | WaitForSingleObject()函数原型及用法 |
|---|---|
| 1 | DWORD WINAPI WaitForSingleObject(//等待单个对象，如果对象置位，则返回 |
| 2 | HANDLE hHandle,//等待的对象的句柄 |
| 3 | DWORD dwMilliseconds//等待超时的时间，如果需要无限等待，设置为 INFINITE |
| 4 | ); |
| 5 | //参考文献：范文庆，周彬彬，安靖. Windows API 开发详解：函数、接口、编程实例[M]. 2013 |
| 6 | 年 1 月北京第 3 次印刷. 北京:人民邮电出版社，2011:216-217. |

5、运行“DiningPhilosophor”目录中的项目程序，观察程序运行结果，查看项目源文件，进一步学习线程的同步与死锁。观察线程间“死锁”时的状态。深入分析死锁产生的原因：假如 5 个哲学家同时饥饿，并且同时拿起左边的筷子，所有的筷子的信号量均为 0。当每个哲学家试图那右边的筷子时，他会永远等待。

6、创建一个“Console”应用程序，在 main()函数中创建 4 个线程，线程的工作就是向屏幕输出几个字符后，就自己结束掉。

（1）程序源码。

| 行数 | Console.cpp |
|---|---|
| 1 | #include <windows.h> |
| 2 | #include <stdio.h> |
| 3 | #define MAX_THREADS 4 |
| 4 | typedef struct _THREAD_PARAM { |
| 5 | DWORD i; |
| 6 | DWORD dwRandom; |
| 7 | DWORD dwData; |
| 8 | } THREAD_PARAM, *LPTHREAD_PARAM; |
| 9 | DWORD WINAPI ThreadProc( LPVOID lpParam ) |

```
{
    printf("TID=%u\n",GetCurrentThreadId());
    return 0;
}
void main()
{
    LPTHREAD_PARAM pData;
    DWORD dwThreadId[MAX_THREADS];
    HANDLE hThread[MAX_THREADS];
    int i;
    // 创建 MAX_THREADS 个线程.
    for( i=0; i<MAX_THREADS; i++ )
    {
        // 创建线程
        hThread[i] = CreateThread(
            NULL,              // 默认安全属性
            0,                 // 默认堆栈大小
            ThreadProc,        // 线程函数
            pData,             // 参数
            0,                 // 默认创建标志
            &dwThreadId[i]);   // 返回 TID
    }
    WaitForMultipleObjects(MAX_THREADS,hThread,TRUE,INFINITE);
    for(i=0;i<MAX_THREADS;i++)
    {
        CloseHandle(hThread[i]);
    }
}
```

（2）程序运行结果截图，举例如图 4、图 5 所示。

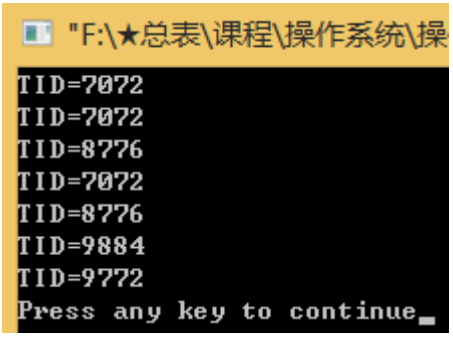


图 4

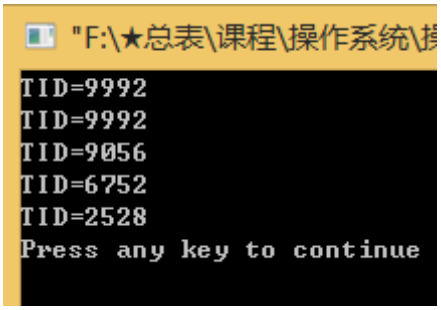


图 5

（3）实验过程与结果分析。

在 main()函数中创建 4 个线程，线程的工作就是向屏幕输出几个字符（线程号）后，就自己结束掉。在主函数生成线程后，线程调用线程函数，进行输出，程序得以实现。