

四川大学计算机学院、软件学院

实 验 报 告

学号：2022141460180 姓名：封欢欢 专业：计算机科学与技术 班级：行政4班 第 10-11 周

课程名称	操作系统课程设计	实验课时	4 小时
实验项目	互斥与同步实验	实验时间	第 10 周到第 11 周
实验目的	1) 回顾经典的同步互斥问题 2) 了解信号量、线程创建等 API 3) 利用 P/V 信号量的编程解决一些经典的同步互斥问题		
实验环境	HUAWEI 电脑 Clion		
实验内容（算法、程序、步骤和方法）	<p>读者写者问题（写者优先）：</p> <p>问题描述： 有一个被许多进程共享的数据区，这个数据区可以是一个文件，或者主存的一块空间，甚至可以是一组处理器寄存器。有一些只读取这个数据区的进程（reader）和一些只往数据区中写数据的进程（writer）。这些读者和写者对数据区的操作必须满足以下条件：读一读允许；读一写互斥；写一写互斥。通过分析可知需要考虑以下几点：</p> <p>（1）任意多的读进程可以同时读这个文件； （2）一次只允许一个写进程往文件中写； （3）如果一个写进程正在往文件中写，禁止任何读进程或写进程访问文件； （4）当有读者在读文件时不允许写者写文件。 （5）写者优先，即当有读者和写者同时等待时，首先满足写者。当一个写者声明想写文件时，不允许新的读者再访问文件。</p> <p>实验方案设计：</p> <p>1. 创建若干线程分别模拟读者操作和写者操作 2. 读线程间和写线程间对各自局部共享资源的访问修改分别采用 RMutex 和 Wmutex 对象，结合 OneWmutex 保证互斥操作 3. 读线程与写线程争用全局临界资源采用 RW_mutex, OneWmutex, mutex。 4. 统管读写线程的线程采用 WaitForMultipleObjects 保证等待所有的线程结束。</p>		

(接上)
实验内容（算法、程序、
步骤和方法）

读取数据文件 data.txt 内容如下：

文件	编辑	查看	
1	R	3	5
2	W	4	5
3	R	5	2
4	R	6	5
5	W	5.1	3

实验代码：

头文件以及线程结构体定义及全局变量定义

```
1  #include<iostream>
2  #include<Windows.h>
3  #include<conio.h>
4  #include<fstream>
5  #include<iostream>
6  using namespace std;
7
8  #define sleep(n) Sleep(n*1000)
9  struct ThreadInfo
10 {
11     int tid;           //线程ID
12     char role;         //扮演角色R or W
13     double delay;      //线程延迟
14     double persist;    //线程读写操作持续时间
15 };
16 int ReadCount = 0;
17 int WriterCount = 0;
18 HANDLE Rmutex;        //用于读者只能一个访问ReadCount
19 HANDLE RW_mutex;      //避免写者与多个读者竞争
20 HANDLE Wmutex;        //用于写者只能有一个访问Writercount
21 HANDLE mutex;         //表示有进程使用书
22 HANDLE OneWmutex;     //只能有一个写者同时写
```

读线程：

```

23 void ReaderThread(LPVOID lpParam)
24 {
25     ThreadInfo* info = (ThreadInfo*)lpParam;
26     sleep(info->delay); //模拟延迟时间
27     printf(format:"Reader thread %d sends the reading require.\n",info->tid);
28
29     WaitForSingleObject(RW_mutex, dwMilliseconds-1);
30     WaitForSingleObject(mutex, dwMilliseconds-1);
31     WaitForSingleObject(Rmutex, dwMilliseconds-1); //对readercount互斥访问
32     if (ReadCount == 0) WaitForSingleObject(OneWmutex, dwMilliseconds-1); //第一位读者申请书,同时防止写者进行操作
33     ReadCount++;
34
35     ReleaseSemaphore(Rmutex, lReleaseCount:1, lpPreviousCount:NULL); //释放互斥信号量
36     ReleaseSemaphore(mutex, lReleaseCount:1, lpPreviousCount:NULL); //释放互斥信号量
37     ReleaseSemaphore(RW_mutex, lReleaseCount:1, lpPreviousCount:NULL); //释放互斥信号量
38
39     printf(format:"Reader thread %d begins to read file.\n", info->tid);
40     sleep(info->persist); //模拟持续时间
41     printf(format:"Reader thread %d finished reading file.\n", info->tid);
42
43     WaitForSingleObject(Rmutex, dwMilliseconds-1); //修改readercount
44     ReadCount--;
45     if (ReadCount == 0) ReleaseSemaphore(OneWmutex, lReleaseCount:1, lpPreviousCount:NULL); //释放资源,写者可写
46     ReleaseSemaphore(Rmutex, lReleaseCount:1, lpPreviousCount:NULL); //释放互斥信号量
47 }

```

写线程:

```

49 void WriterThread(LPVOID lpParam) {
50     ThreadInfo* info = (ThreadInfo*)lpParam;
51     sleep(n:info->delay);
52     printf(format:"Writer thread %d sends the writing require.\n",info->tid);
53     WaitForSingleObject(Wmutex, dwMilliseconds-1); //对writercount互斥访问
54     if (WriterCount == 0) WaitForSingleObject(mutex, dwMilliseconds-1); //第一位写者申请资源
55     WriterCount++;
56     ReleaseSemaphore(Wmutex, lReleaseCount:1, lpPreviousCount:NULL); //释放资源
57
58     WaitForSingleObject(OneWmutex, dwMilliseconds-1);
59     printf(format:"Writer thread %d begins to write to the file.\n", info->tid);
60     sleep(n:info->persist); //模拟持续时间
61     printf(format:"Writer thread %d finished writing to the file.\n", info->tid);
62     ReleaseSemaphore(OneWmutex, lReleaseCount:1, lpPreviousCount:NULL); //释放资源
63
64     WaitForSingleObject(Wmutex, dwMilliseconds-1); //对writercount互斥访问
65     WriterCount--;
66     if (WriterCount == 0) ReleaseSemaphore(mutex, lReleaseCount:1, lpPreviousCount:NULL); //释放资源
67     ReleaseSemaphore(Wmutex, lReleaseCount:1, lpPreviousCount:NULL);
68 }

```

Main 函数:

```

Writer Priority:

Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the writing require.
Reader thread 3 sends the reading require.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 1 finished reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Reader thread 3 begins to read file.
Reader thread 3 finished reading file.
Reader thread 4 begins to read file.
Reader thread 4 finished reading file.

Allreader and eriter have finished operating !
|

```

```

//读者线程
for (int i = 0; i < n_thread; i++)
{
    if((thread_info[i].sig=='R')||thread_info[i].role=='e')
    {
        h_thread[i]=CreateThread(NULL,0,NULL,(LPVOID) ReaderThread,&thread_info[i],0,&thread_ID);
    }
    else
    {
        h_thread[i] = CreateThread(NULL,0,NULL,(LPVOID) WriterThread,&thread_info[i],0,&thread_ID);
    }
}
//等待所有线程结束
DWORD wait_for_all=WaitForMultipleObjects(nCount,h_thread,true,INFINITE);
cout<<endl;
printf("Reader and writer have finished operating!\n");
_getch();
return 0;
}

```

输出结果:

```

Reader Priority:
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the writing require.
Reader thread 3 sends the reading require.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 1 finished reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Reader thread 3 begins to read file.
Reader thread 3 finished reading file.
Reader thread 4 begins to read file.
Reader thread 4 finished reading file.

Allreader and eriter have finished operating !

```

消费者生产者问题:

问题描述:

生产者线程生产出产品以后, 消费者线程去消费产品; 当消费者线程去消费产品时, 发现还没有产品生产出来, 就等待, 等生产者线程生产出产品以后, 消费者线程才能继续往下执行。

需要考虑的点:

- (1) 生产者可以同时生产, 但是向仓库放的时候互斥的放
- (2) 在仓库容量为 0 (即满的时候) 的时候, 生产者不能向仓库放, 需等待消费者消费后才能放。
- (3) 消费者从仓库去货的时候互斥, 即同一时间只能一个消费者访问仓库。
- (4) 在仓库内没有货物的时候, 消费者需等待生产者生产出商品后才能继续消费。

读取数据文件 data2.txt 内容如下:

文件	编辑	查看	
1	P	3	5
2	C	4	5
3	P	5	2
4	P	6	5
5	C	5.1	3

实验代码：

头文件以及线程结构体定义及全局变量定义：

```

1  #include<iostream>
2  #include<Windows.h>
3  #include<conio.h>
4  #include<fstream>
5  #include<iostream>
6  using namespace std;
7
8  #define sleep(n) Sleep(n*1000)
9  struct ThreadInfo
10 {
11     int tid;           //线程ID
12     char role;         //扮演角色R or W
13     double delay;      //线程延迟
14     double persist;    //线程读写操作持续时间
15 };
16 HANDLE Empty;
17 HANDLE mutex;
18 HANDLE Full;

```

生产线程：

```

void ProducersThread(LPVOID lpParam)
{
    ThreadInfo* info = (ThreadInfo*)lpParam;
    sleep(info->delay); //模拟延时时间
    printf(format:"Producer thread %d sends the producing require.\n",info->tid);
    printf(format:"Producer thread %d begins to product item.\n", info->tid);
    sleep(info->persist); // 模拟持续时间
    printf(format:"Producer thread %d finished producing item.\n", info->tid);
    WaitForSingleObject(Empty, dwMilliseconds:-1);
    WaitForSingleObject(mutex, dwMilliseconds:-1); //对传送带互斥访问
    printf(format:"Producer thread %d had set the item.\n", info->tid);
    ReleaseSemaphore(mutex, |ReleaseCount:1, |lpPreviousCount:NULL);
    ReleaseSemaphore(Full, |ReleaseCount:1, |lpPreviousCount:NULL);
}

```

消费线程：

```

34 void ConsumersThread(LPVOID lpParam) {
35     ThreadInfo* info = (ThreadInfo*)lpParam;
36     sleep(info->delay);
37     printf(format:"Consumer thread %d sends the consuming require.\n",info->tid);
38     WaitForSingleObject(Full, dwMilliseconds:-1);
39     WaitForSingleObject(mutex, dwMilliseconds:-1);
40     ReleaseSemaphore(mutex, lReleaseCount:1, lpPreviousCount:NULL); //释放资源
41     ReleaseSemaphore(Full, lReleaseCount:1, lpPreviousCount:NULL); //释放资源
42     printf(format:"Writer thread %d begins to consume the item.\n", info->tid);
43     sleep(info->persist); // 模拟持续时间
44     printf(format:"Writer thread %d finished consuming the file.\n", info->tid);
45 }

```

Main 函数:

```

46
47 int main()
48 {
49     DWORD n_thread=0; //线程数目
50     DWORD thread_ID; //线程ID
51     //线程对象数组
52     HANDLE h_thread[20];
53     ThreadInfo thread_info[20];
54     //创建互斥量
55     Empty=CreateSemaphore(NULL, lInitialCount:10, lMaximumCount:10, lpName:LPCSTR("mutex_for_readcount"));
56     Full = CreateSemaphore(NULL, lInitialCount:0, lMaximumCount:10, lpName:"mutex_for_writercount");
57     mutex = CreateSemaphore(NULL, lInitialCount:1, lMaximumCount:1, lpName:NULL); //
58     //读取输入文件
59     cout<<"Producers and Consumers questions:"<<endl;
60     cout<<endl;
61     ifstream inFile;
62     inFile.open("data2.txt");
63     if(!inFile) {
64         printf(format:"erroe in open file!\n");
65         return -1;
66     }
67     while(inFile>>thread_info[n_thread].tid) {
68         inFile>>thread_info[n_thread].role;
69         inFile>>thread_info[n_thread].delay;
70         inFile>>thread_info[n_thread].persist;
71         inFile.get();
72         n_thread++;
73     }

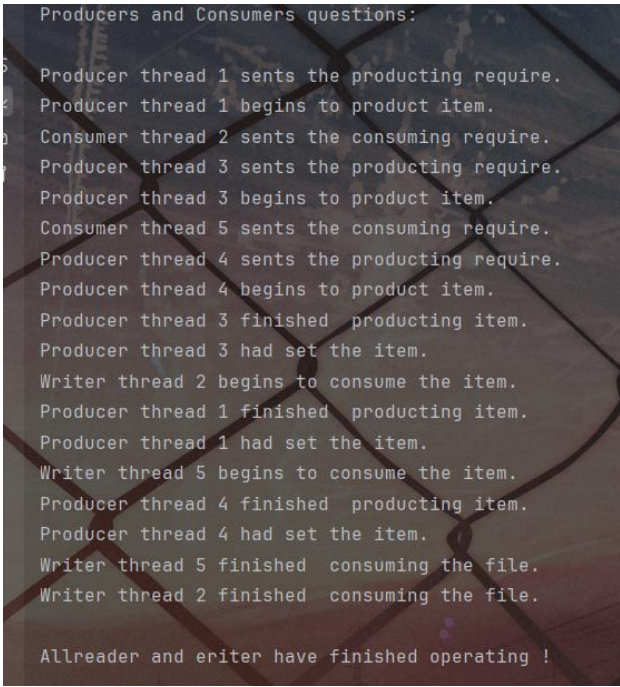
```

```

74 //创建线程
75 for (int i = 0; i < n_thread; i++)
76 {
77     if(thread_info[i].role=='P'){thread_info[i].role='p'}
78     {
79         h_thread[i]=CreateThread(lpThreadAttributes:NULL, dwStackSize:0, (LPTHREAD_START_ROUTINE)ProducersThread,&thread_info[i], dwCreationFlags:0,&thread_ID);
80     }
81     else
82     {
83         h_thread[i] = CreateThread(lpThreadAttributes:NULL, dwStackSize:0, (LPTHREAD_START_ROUTINE)ConsumersThread,&thread_info[i], dwCreationFlags:0,&thread_ID);
84     }
85 }
86 //等待所有线程结束
87 DWORD result=WaitForMultipleObjects(n_thread, h_thread, TRUE, INFINITE);
88 cout<<endl;
89 printf(format:"Allreader and eriter have finished operating \n");
90 getch();
91 return 0;

```

输出结果:

	 <pre>Producers and Consumers questions: 5 6 Producer thread 1 sends the producting require. 7 Producer thread 1 begins to product item. 8 Consumer thread 2 sends the consuming require. 9 Producer thread 3 sends the producting require. 10 Producer thread 3 begins to product item. 11 Consumer thread 5 sends the consuming require. 12 Producer thread 4 sends the producting require. 13 Producer thread 4 begins to product item. 14 Producer thread 3 finished producting item. 15 Producer thread 3 had set the item. 16 Writer thread 2 begins to consume the item. 17 Producer thread 1 finished producting item. 18 Producer thread 1 had set the item. 19 Writer thread 5 begins to consume the item. 20 Producer thread 4 finished producting item. 21 Producer thread 4 had set the item. 22 Writer thread 5 finished consuming the file. 23 Writer thread 2 finished consuming the file. Allreader and eriter have finished operating !</pre>
数据记录 和计算	无数据记录
结 论 (结 果)	<p>读者写者问题:</p> <p>这个问题根据优先级不同,有不同的线程执行顺序,在以读者优先的时候,就算写者发送了请求,也会被阻塞,读者线程全部结束才会解锁,以写者优先的话则相反,但是已经进行的进程会让它先执行完,以免产生死锁。</p> <p>生产者消费者问题:</p> <p>这个问题也是可以分为很多种,比如单生产者-单消费者,单生产者-多消费者,多生产者-单消费者,多生产者-多消费者。之间有很小的差异,实现方式也有多种,实现的时候要考虑到生产者和消费者之间的前驱后驱关系,以及他们各自之间的互斥关系。</p>
小 结	<p>4) 通过此次实验,我回顾经典的同步互斥问题,了解了信号量、线程创建等 API,也亲手利用 P/V 信号量的编程解决了一些经典的同步互斥问题。在编程的过程中,发生了过或大或小的错误,多次造成了死锁现象,使我更加亲切的体会,理解了死锁。当然在编程的过程中我对经典同步互斥问题的理解也更进一步加深,也是不禁赞叹前人抽象总结出来的问题以及其解决方法的精妙。</p>

指导老师评议	成绩评定： 指导教师签名：
--------	----------------------