

线程篇 (下)



日成蝶—Windows API 编程入门

七日做茧，一朝成蝶！



主讲：袁春旭

个人博客：<http://8413723.blog.51cto.com/>

课程主页：<http://edu.51cto.com/lecturer/8403723.html>

原子访问-原子操作

线程同步

线程同步解决什么问题？

问题一：不同线程函数的执行必须有先后顺序。

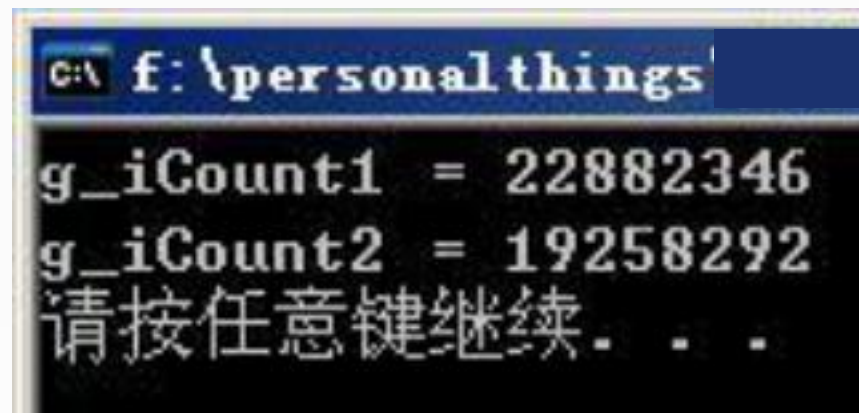
问题二：同一线程函数中一段代码必须作为一个单元执行。

线程同步-案例二

```
//定义两个全局变量，用来计数
int g_iCount1 = 0;
int g_iCount2 = 0;
//定义一个全局变量，用来控制while循环
int g_iFlag = 1;
//下面函数不断让变量g_iCount1和g_iCount2自加1
DWORD WINAPI ThreadProc(LPVOID lpParameter) {
    while(g_iFlag) {
        g_iCount1++; ==> g_iCount1 = g_iCount1 + 1;
        g_iCount2++; ==> g_iCount2 = g_iCount2 + 1;
    }
    return 0;
}
```

线程同步-案例二

```
int main(void)
{
    HANDLE hThread[2];
    hThread[0] = CreateThread(NULL, NULL, ThreadProc, NULL, 0, NULL);
    hThread[1] = CreateThread(NULL, NULL, ThreadProc, NULL, 0, NULL);
    Sleep(100);
    //修改全局标志位使两个线程停止运行
    g_iFlag = 0;
    printf("g_iCount1 = %d\n", g_iCount1);
    printf("g_iCount2 = %d\n", g_iCount2);
    CloseHandle(hThread[0]);
    CloseHandle(hThread[1]);
    return 0;
}
```



```
c:\ f: \personalthings
g_iCount1 = 22882346
g_iCount2 = 19258292
请按任意键继续. . .
```

线程同步-案例二

线程一

```
g_iCount1 = g_iCount1 + 1;
```

读取g_iCount1到临时位置A

1

A+1赋值给A

3

将A的值赋值给g_iCount1

4

线程二

```
g_iCount1 = g_iCount1 + 1;
```

读取g_iCount1到临时位置A

2

A+1赋值给A

5

将A的值赋值给g_iCount1

6

InterLocked系列函数

InterLocked系列函数

LONG InterlockedIncrement(LONG volatile* Addend)

LONGLONG InterlockedIncrement64(LONGLONG volatile* Addend)

Addend：原子递增变量的指针。

返回值：变量递增后的值。

线程同步-案例二

```
//定义两个全局变量，用来计数
int g_iCount1 = 0;
int g_iCount2 = 0;
//定义一个全局变量，用来控制while循环
int g_iFlag = 1;
//下面函数不断让变量g_iCount1和g_iCount2自加1
DWORD WINAPI ThreadProc(LPVOID lpParameter) {
    while(g_iFlag) {
        InterlockedIncrement((LONG*)&g_iCount1);
        InterlockedIncrement((LONG*)&g_iCount2);
    }
    return 0;
}
```

InterLocked系列函数

LONG InterlockedDecrement(LONG volatile* Addend)

LONGLONG InterlockedDecrement64(LONGLONG volatile* Addend)

Addend：原子递减变量的指针。

返回值：变量递减后的值。

InterLocked系列函数

LONG InterlockedExchangeAdd(LONG volatile* Addend, LONG Value)

LONGLONG InterlockedExchangeAdd64(
LONGLONG volatile* Addend,
LONGLONG Value)

Addend : 原子加法变量的指针。

Value : 增量。

返回值 : 变量变化后的值。

思考 : 如何做原子减法 ?

InterLocked系列函数

LONG InterlockedExchange(LONG volatile* Addend, LONG Value)

LONGLONG InterlockedExchange64(
LONGLONG volatile* Addend,
LONGLONG Value)

Addend：原子操作变量的指针。

Value：给Addend赋值的值。

返回值：变量**变化前**的值。

InterLocked系列函数

```
PVOID InterlockedExchangePointer(PVOID volatile *Target, PVOID Value);
```

Target：二重指针，需要替换的二重指针的值。

Value：用该地址值替换Target中的地址。

返回值：Target在被替换前的地址值。

InterLocked系列函数

```
LONG InterlockedCompareExchange(  
    LONG* Destination,  
    LONG ExChange,  
    LONG Comperand  
);
```

Destination：需要替换的变量地址。

ExChange：给Destination赋值的值。

Comperand：需要比较的值，如果与Destination相同，则替换。

返回值：**变化前**的值。

InterLocked系列函数

```
LONGLONG InterlockedCompareExchange64 (  
    LONGLONG        volatile *Destination,  
    LONGLONG        Exchange,  
    LONGLONG        Comparand  
);
```

Destination：需要替换的变量地址。

ExChange：给Destination赋值的值。

Comperand：需要比较的值，如果与Destination相同，则替换。

返回值：**变化前**的值。

InterLocked系列函数

```
PVOID InterlockedCompareExchangePointer(  
    PVOID *Destination ,  
    PVOID Exchange ,  
    PVOID Comparand  
);
```

Destination : 需要替换的指针的地址。

ExChange : 给Destination赋值的值。

Comperand : 需要比较的值，如果与Destination相同，则替换。

返回值 : **变化前**的值。

InterLocked系列函数

```
SHORT InterlockedAnd16(SHORT volatile *Destination, SHORT Value);
```

```
LONG InterlockedAnd(LONG volatile *Destination, LONG Value);
```

```
LONGLONG InterlockedAnd64(LONGLONG volatile *Destination, LONGLONG Value);
```

Destination：与Value做按位与运算，并保存结果。

Value：与Destination做按位与运算的值。

返回值：**变化前**的值。

InterLocked系列函数

```
SHORT InterlockedOr16(SHORT volatile *Destination, SHORT Value);
```

```
LONG InterlockedOr(LONG volatile *Destination, LONG Value);
```

```
LONGLONG InterlockedOr64(LONGLONG volatile *Destination, LONGLONG Value);
```

Destination：与Value做按位或运算，并保存结果。

Value：与Destination做按位或运算的值。

返回值：**变化前**的值。

InterLocked系列函数

```
SHORT InterlockedXor16(SHORT volatile *Destination, SHORT Value);
```

```
LONG InterlockedXor(LONG volatile *Destination, LONG Value);
```

```
LONGLONG InterlockedXor64(LONGLONG volatile *Destination, LONGLONG Value);
```

Destination：与Value做按位异或运算，并保存结果。

Value：与Destination做按位异或运算的值。

返回值：**变化前**的值。

编码实战



Thank You !