








网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

第4章 进程管理

-  4.1进程概念
-  4.2进程控制
-  4.3线程
-  4.4临界区和锁
-  4.5同步和P-V操作
-  4.6Windows和Linux同步机制
-  4.7进程通信



华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

4.6 Windows和Linux同步机制



4.6.1 Windows同步机制



4.6.2 Linux父子进程同步

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

《操作系统原理》

4.6.1 Windows同步机制

教师：苏曙光

华中科技大学软件学院

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

Windows进程同步机制

■ 临界区（锁） ✓

- CRITICAL_SECTION

■ 互斥量(Mutex)（锁） ✓

- HANDLE

■ 信号量（Semaphore） -

- HANDLE

■ 事件（Event） ✓

- HANDLE

■ 等待操作

- WaitForSingleObject和WaitForMultipleObjects

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

Windows同步机制

■ 临界区 (CRITICAL_SECTION)

- 在进程内使用，保证仅一个线程可以申请到该对象。
- 临界区内是临界资源的访问

□ 相关的API函数：

- InitializeCriticalSection();
- 初始化临界区
- DeleteCriticalSection();
- 删除临界区
- EnterCriticalSection();
- LeaveCriticalSection();
- 退出临界区(开锁)。

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

用3个线程共同把nSum累加到240.

```
1 //全局变量
2 int nSum = 0;
3 int NUMBER = 80;
4 //累加线程函数 Accumulate
5 DWORD WINAPI Accumulate(LPVOID lpParam)
6 {
7     for(int i=0; i<NUMBER; i++)
8     {
9         int iCopy = nSum;
10         nSum = iCopy+1;
11     }
12     return 0;
13 }
14 int main(int argc, char* argv[])
15 {
16     hThread[0] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
17     hThread[1] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
18     hThread[2] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
19     WaitForMultipleObjects(3, hThread, TRUE, INFINITE);
20     printf("    nSum = %d", nSum);
21     return 0;
22 }
```

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

等待函数WaitForXXXObject

功能：等待目标对象变成有信号的状态就返回。



DWORD WaitForMultipleObjects (

DWORD nCount, //等待的目标对象的数量

CONST HANDLE *lpHandles, //目标对象的句柄数组

BOOL fWaitAll, //等待方式

DWORD dwMilliseconds); // 等待时间，单位MS



DWORD WaitForSingleObject (

HANDLE hHandle, //等待的目标对象的句柄

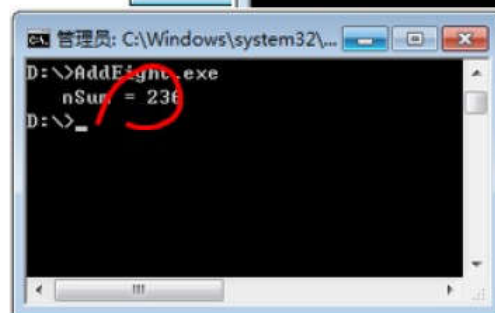
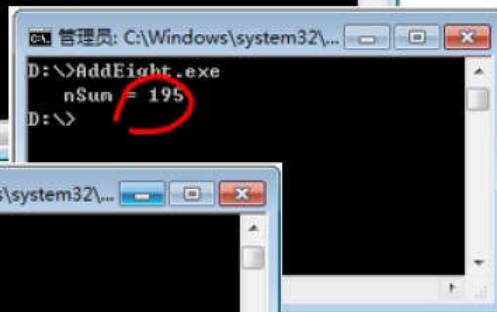
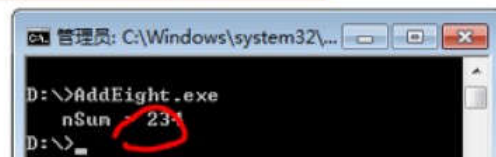
DWORD dwMilliseconds); // 等待时间，单位MS

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

用3个线程共同把nSum累加到240.

```
1 //全局变量
2 int nSum = 0;
3 int NUMBER = 80 ;
4 //累加线程函数 Accumulate
5 DWORD WINAPI Accumulate(LPVOID lpParam)
6 {
7     for(int i=0;i<NUMBER; i++)
8     {
9         int iCopy = nSum;
10        nSum = iCopy+1;
11    }
12    return 0;
13 }
14 int main(int argc, char* argv[])
15 {
16     hThread[0] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
17     hThread[1] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
18     hThread[2] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
19     WaitForMultipleObjects(3,hThread,TRUE,INFINITE);
20     printf("    nSum = %d",nSum);
21     return 0;
22 }
```

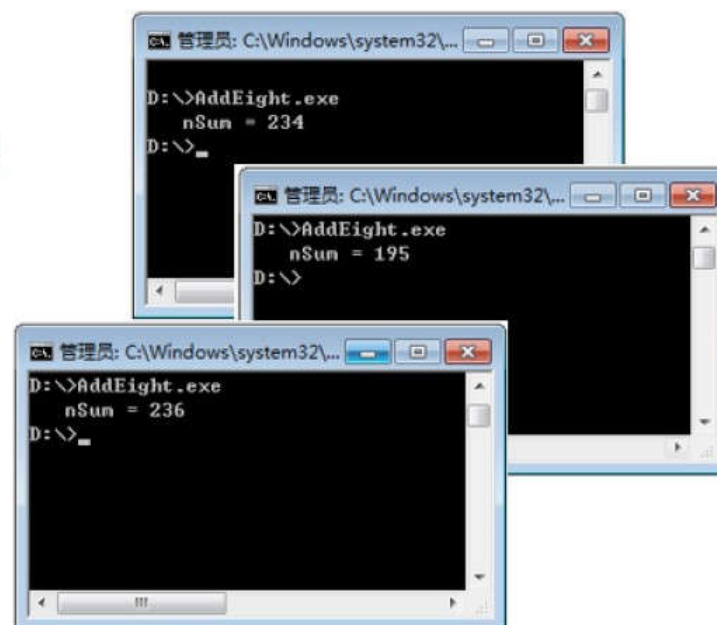


华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

```
1 //全局变量
2 int nSum = 0;
3 int NUMBER = 80 ;
4 //累加线程函数 Accumulate
5 DWORD WINAPI Accumulate(LPVOID lpParam)
6 {
7     for(int i=0;i<NUMBER; i++)
8     {
9         int iCopy = nSum;
10        nSum = iCopy+1;
11    }
12    return 0;
13 }
14 int main(int argc, char* argv[])
15 {
16     hThread[0] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
17     hThread[1] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
18     hThread[2] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
19     WaitForMultipleObjects(3,hThread,TRUE,INFINITE);
20     printf("    nSum = %d",nSum);
21     return 0;
22 }
```

多个线程
同时就绪



华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址: www.icourses.cn

主页搜索“苏曙光”即可进入MOOC课堂

```
1 //全局变量
2 int nSum = 0;
3 int NUMBER = 80;
4 CRITICAL_SECTION cs; //定义临界区对象
5 //累加线程函数 Accumulate
6 DWORD WINAPI Accumulate(LPVOID lpParam)
7 {
8     for(int i=0;i<NUMBER;i++)
9     {
10         EnterCriticalSection(&cs); //进入临界区
11         int iCopy = nSum;
12         nSum = iCopy+1;
13         LeaveCriticalSection(&cs); //退出临界区
14     }
15     return 0;
16 }
17
18 int main(int argc, char* argv[])
19 {
20     InitializeCriticalSection(&cs); //初始化临界区对象cs
21     HANDLE hThread[3]; //定义线程句柄, 将创建3个线程
22     hThread[0] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
23     hThread[1] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
24     hThread[2] = CreateThread(NULL, 0, Accumulate, NULL, 0, NULL);
25     WaitForMultipleObjects(3, hThread, TRUE, INFINITE);
26     printf("    nSum = %d", nSum);
27     return 0;
28 }
```

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂



互斥量(Mutex)

- 保证只有**一个**线程或进程可以申请到该对象。
- 可以**跨进程**使用
- 可以有**名称**。
- 互斥量比临界区要耗费更多资源，速度慢。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

■ 用在互斥量上的API函数：

- **HANDLE CreateMutex(** //创建互斥量
LPSECURITY_ATTRIBUTES lpMutexAttributes,
BOOL bInitialOwner, // 初始化互斥量的状态：真或假
LPCTSTR lpName // 名字，可为NULL但不能跨进程用);
- **HANDLE OpenMutex(**//打开一个存在的互斥量
DWORD dwDesiredAccess,
BOOL bInheritHandle,
LPCTSTR lpName // 名字);
- **BOOL ReleaseMutex(** HANDLE hMutex);
- **BOOL CloseHandle(**//关闭互斥量
HANDLE hObject // 句柄);

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂



信号量 (Semaphore)

- 允许指定数目的多个线程/进程访问临界区。''
- 一种资源计数器，用于限制并发线程的数量。
- 初始值可设为N，则表示允许N个进程/线程**并发访问资源**。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

■ 用于信号量操作的API函数

- **HANDLE CreateSemaphore**(//创建信号量
LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, // 安全属性
LONG lInitialCount, // 初始值
LONG lMaximumCount, // 最大值
LPCTSTR lpName // **名字**);
- **HANDLE OpenSemaphore**(//打开信号量
DWORD dwDesiredAccess, // 存取方式
BOOL bInheritHandle, // 是否能被继承
LPCTSTR lpName // **名字**);
- **BOOL ReleaseSemaphore**(//释放信号量
HANDLE hSemaphore, // 句柄
LONG **lReleaseCount**, // **释放数，让信号量的值增加的数量**
LPLONG lpPreviousCount // 得到释放前信号量的值，可为NULL);
- **BOOL CloseHandle**(//关闭信号量
HANDLE hObject // **句柄**);

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

■ 信号量 (Semaphore)

□ 信号量的值可以通过相应函数增或减。

□ WaitForSingleObject将信号量减1

□ ReleaseSemaphore将信号量增1。

■ 信号状态

□ 信号量的值大于0时，有信号状态。

□ 信号量的值小于等于0时，为无信号状态

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

使用信号量同步的例子

//in main function

```
{    //创建信号灯
    HANDLE hSU=NULL;
    hSU = CreateSemaphore(NULL,2,2,'SU');
    HANDLE hThread [3]; //创建3个线程
    CWinThread* pT1=AfxBeginThread(AccessDB, (void*)1 );
    CWinThread* pT2=AfxBeginThread(AccessDB, (void*)2 );
    CWinThread* pT3=AfxBeginThread(AccessDB, (void*)3 );
    hThread[0]=pT1->m_hThread;
    hThread[1]=pT2->m_hThread;
    hThread[2]=pT3->m_hThread;
    WaitForMultipleObjects(3, hThread, TRUE,INFINITE);
    CloseHandle(hSU); //关闭句柄
}
```

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

使用信号量同步的例子

//功能：信号量初始值为2，保证最多只有2个线程可以同时进行数据库访问。

//hSU = CreateSemaphore(NULL,2,2,"SU");

DWORD AccessDB(void* pD) //线程函数

{

HANDLE hSU = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "SU");

while (TRUE)

{

//此处是模拟数据库访问！

printf("Do database Access ! \n");

Sleep(100);

}

return 0;

}

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

■ 事件（Event）

- 用于通知一个或多个线程某事件出现或标识某操作已经完成。

■ 事件对象的分类

- 自动重置的事件：使用WaitForSingleObject等待到事件对象变为**有信号状态**后该事件对象**自动**变为**无信号状态**。
- 人工重置的事件：使用WaitForSingleObject等待到事件对象变为**有信号状态**后该事件对象的状态**不变**，除非人工重置事件。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂



事件相关的API函数

- **HANDLE CreateEvent** (//创建事件对象
LPSECURITY_ATTRIBUTES lpEventAttributes, // 安全属性
BOOL bManualReset, // 是否为人工重置
BOOL bInitialState, // 初始状态是否为有信号状态
LPCTSTR lpName // 名字);
- **HANDLE OpenEvent** (//打开事件对象
DWORD dwDesiredAccess, // 存取方式
BOOL bInheritHandle, // 是否能够被继承
LPCTSTR lpName // 名字);
- **BOOL ResetEvent** (//设置事件为无信号状态
HANDLE hEvent // 句柄);
- **BOOL SetEvent** (//设置事件有信号状态
HANDLE hEvent // 句柄);
- **BOOL CloseHandle** (//关闭事件对象
HANDLE hObject // 句柄);

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

Windows同步机制

■ 临界区对象

EnterCriticalSection(); ← **P操作**
LeaveCriticalSection(); ← **V操作**

■ 互斥量对象

ReleaseMutex(); ← **V操作**

■ 事件对象

BOOL SetEvent(); ← **V操作**

■ 信号量对象

CreateSemaphore(); //创建一个信号量并初始化

ReleaseSemaphore(); ← **V操作**

■ 等待机制

WaitForSingleObject(); ← **P操作**

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有