# 学习大纲

|  |
| --- |
|  |
|  |

## 为什么要进行线程同步？

|  |
| --- |
| **线程有可能和其他线程共享一些资源，比如，内存，文件，数据库等。 当多个线程同时读写同一份共享资源的时候，可能会引起冲突。这时候，我们需要引入线程“同步”机制，即各位线程之间要有个先来后到，不能一窝蜂挤上去抢作一团。** [**线程同步**](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%90%8C%E6%AD%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvPhN-nWKbrjm1rymsuHRY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1cLPW6dP1c1)**的真实意思和字面意思恰好相反。**[**线程同步**](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%90%8C%E6%AD%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvPhN-nWKbrjm1rymsuHRY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1cLPW6dP1c1)**的真实意思，其实是“排队”：几个线程之间要排队，一个一个对共享资源进行操作，而不是同时进行操作。** [**线程同步**](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%90%8C%E6%AD%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvPhN-nWKbrjm1rymsuHRY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1cLPW6dP1c1)**的方法 (1)wait():使一个线程处于等待状态，并且释放所持有的对象的lock。 (2)sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉   InterruptedException异常。 (3)notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的  唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。 (4)notityAll ():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，  而是让它们竞争**  因此**线程同步**就非常重要，常用的线程同步方式就是**加锁**。锁是一种非强制机制，每一个线程在访问数据或资源之前都应该获取锁（或者说上锁、lock），并在访问结束之后释放锁(或者说解锁、unlock)。在锁被别的线程占用时，当前线程就需要等待也即是阻塞，直到锁被释放，然后当前线程抢到锁了，才能继续执行。 |

## 线程同步的方法

## 1.临界区，参考：[C++多线程编程：多线程同步之临界区 CriticalSection](C++多线程编程：多线程同步之临界区%20CriticalSection.docx)

## 2.互斥量，参考：[C++多线程编程：同步之互斥量Mutex](C++多线程编程：同步之互斥量Mutex.docx)

## 3.信号量，参考：[C++多线程编程：同步之信号量 Semaphore](C++多线程编程：同步之信号量%20Semaphore.docx)

## 4.事件， 参考：[C++多线程编程：同步之事件Event](C++多线程编程：同步之事件Event.docx)

# 演练

## 1.新建一个MFC基于对话框的应用程序，取名Lesson56-thread-sychronize

|  |
| --- |
|  |

## 2. 这里我们使用另外一种方法来解决按下回传程序退出的问题，先在对话框的头文件里面定义这个函数

|  |
| --- |
|  |

## 3.然后在对话框的cpp文件里面实现他的代码

|  |
| --- |
|  |

## 4.然后就可以把对话框默认的所有控件删除，

|  |
| --- |
|  |

## 5.把对话框的高度调整小一点，然后添加一个按钮和一个静态文本

|  |
| --- |
|  |

## 6.点击按钮进入他的点击处理事件代码，我们想在这里创建一个新的线程，所以需要先在头文件里面定义一个全局线程函数

|  |
| --- |
|  |

## 7.然后在对话框的cpp添加对这个函数的实现，我们需要先定义两个多线程共享的变量

|  |
| --- |
|  |

## 8.然后我们完成线程函数的编写

|  |
| --- |
|  |

## 9.然后我们回到按钮的点击事件代码，创建一个线程

|  |
| --- |
|  |

### 此时编译运行程序，很快就有结果，如果只有一个辅助线程，没有什么问题，不存在争抢共享资源的问题

## 10.然后我们再添加一个按钮，把静态文本拉长，然后设置文本居中，

|  |
| --- |
|  |

## 11.然后我们再创建一个线程函数

|  |
| --- |
|  |
|  |

## 12.在第二个按钮的点击事件函数里面又创建一个线程

|  |
| --- |
|  |

### 编译运行程序，此时如果我先点击按钮1，等到这个线程执行完成后，再点击按钮2.最终的数字是200000000

|  |
| --- |
|  |

### 但是如果我点击按钮后，很快有点击按钮2.最终的结果就是不正确的

|  |
| --- |
|  |

## 第57课笔记

## 13.解决这个问题的办法就是使用线程同步，第一种方式就是使用临界值，给容易出错的地方加锁，执行完后需要解锁，先定义一个临界取类的全局对象

|  |
| --- |
|  |

## 14.在对话框类的构造函数里面创建一般临界区对象赋值给这个指针

|  |
| --- |
|  |

## 15.然后给对话框类定义析构函数，在析构函数里面删除这个指针变量

|  |
| --- |
|  |

## 16.然后在cpp文件中定义析构函数，在里面删除对象并且把指针赋值为空

|  |
| --- |
|  |

## 17.然后我们在两个线程函数的关键地方加锁，运行完成后解锁

|  |
| --- |
| //定义全局线程函数  UINT ThreadProc1(LPVOID lpParam)  {  for(int i=1;i<=100000000;i++)  {  //加锁  gpcs->Lock();  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  gpcs->Unlock();  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  UINT ThreadProc2(LPVOID lpParam)  {  for(int i=1;i<=100000000;i++)  {  //加锁  gpcs->Lock();  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  gpcs->Unlock();  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  } |

### 编译运行程序，此时不管是方便点击还是一起点击的结果都是正确的

|  |
| --- |
|  |

### 注意：使用了临界区对象后，计算结果是对的，但是速度比较慢，没有问题，因为我们必须保证我们的程序的运行结果是对的，否则程序就没有意义。

## 18，我们把临界区对象的代码注释了，接着我们来学习CMutex对象的使用.先定义一个CMutex的全局指针变量，然后在构造函数里面给他赋值，在析构函数里面删除他指向的对象并且把他设置为空

|  |
| --- |
|  |

## 19，然后在代码关键区创建锁，执行完毕后解锁

|  |
| --- |
| //使用互斥量的方式  UINT ThreadProc1(LPVOID lpParam)  {  CSingleLock sLock(gpmutex);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  UINT ThreadProc2(LPVOID lpParam)  {    CSingleLock sLock(gpmutex);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  } |

### 使用互斥量对象非常的慢，因为它是内核对象

## 20.我们来看看信号量，也是需要定义全局指针变量，然后在对话框的构造函数里面赋值，在析构函数里面删除对象，设置指针为空，其实他的代码和互斥量是一样的。改一下指针变量即可

|  |
| --- |
|  |
|  |
| //使用信号量的方式  UINT ThreadProc1(LPVOID lpParam)  {  CSingleLock sLock(gpsmph);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  UINT ThreadProc2(LPVOID lpParam)  {    CSingleLock sLock(gpsmph);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  } |

### 信号量也是很慢

## 21.下面我们来学习事件对象的方法，首先在对话框cpp文件里面定义一个全局事件对象，然后在对话框的构造函数里面给他赋值，然后在析构函数里面删除它指向的对象并且把它设置为0

|  |
| --- |
|  |
|  |
| //使用事件对象的方式  UINT ThreadProc1(LPVOID lpParam)  {  CSingleLock sLock(gpevent);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  //使用事件对象的方式，解锁后需要设置事件，也就是发信号说资源可以用了  gpevent->SetEvent();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  UINT ThreadProc2(LPVOID lpParam)  {    CSingleLock sLock(gpevent);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  //使用事件对象的方式，解锁后需要设置事件，也就是发信号说资源可以用了  gpevent->SetEvent();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  } |

## 这个event对象也是很慢

## 这一节的学习到此为止，完整代码如下，注意这里把56和57一起做了，因为56没有然后线程同步代码

## Lesson56-thread-sychronizeDlg.h

|  |
| --- |
| // Lesson56-thread-sychronizeDlg.h : 头文件  //  #pragma once  //声明全局线程函数  UINT ThreadProc1(LPVOID lpvoid);  UINT ThreadProc2(LPVOID lpvoid);  // CLesson56threadsychronizeDlg 对话框  class CLesson56threadsychronizeDlg : public CDialogEx  {  // 构造  public:  CLesson56threadsychronizeDlg(CWnd\* pParent = NULL); // 标准构造函数  ~CLesson56threadsychronizeDlg();//声明析构函数  // 对话框数据  enum { IDD = IDD\_LESSON56THREADSYCHRONIZE\_DIALOG };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  //声明防止按下车就退出程序的函数  protected:  virtual BOOL PreTranslateMessage(MSG\* pMsg);  // 实现  protected:  HICON m\_hIcon;  // 生成的消息映射函数  virtual BOOL OnInitDialog();  afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);  afx\_msg void OnPaint();  afx\_msg HCURSOR OnQueryDragIcon();  DECLARE\_MESSAGE\_MAP()  public:    afx\_msg void OnBnClickedButton1();  afx\_msg void OnBnClickedButton2();  }; |

## Lesson56-thread-sychronizeDlg.cpp

|  |
| --- |
| // Lesson56-thread-sychronizeDlg.cpp : 实现文件  //  #include "stdafx.h"  #include "Lesson56-thread-sychronize.h"  #include "Lesson56-thread-sychronizeDlg.h"  #include "afxdialogex.h"  #ifdef \_DEBUG  #define new DEBUG\_NEW  #endif  //先定义两个多线程共享的变量  int k=1;  int total = 0;  CCriticalSection \*gpcs;//临界区指针  CMutex \*gpmutex;//互斥量指针  CSemaphore \*gpsmph; //信号量指针  CEvent \*gpevent; //事件对象指针  //定义全局线程函数  //使用临界区对象的解决方式  //UINT ThreadProc1(LPVOID lpParam)  //{  // for(int i=1;i<=100000000;i++)  // {  // //加锁  // gpcs->Lock();  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // gpcs->Unlock();  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  //  //UINT ThreadProc2(LPVOID lpParam)  //{  // for(int i=1;i<=100000000;i++)  // {  // //加锁  // gpcs->Lock();  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // gpcs->Unlock();  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  ////使用互斥量的方式  //UINT ThreadProc1(LPVOID lpParam)  //{  // CSingleLock sLock(gpmutex);//用互斥量对象创建一把锁  // for(int i=1;i<=100000000;i++)  // {  // //加锁，这种互斥锁有可能没有锁住，需要做个判断  // sLock.Lock();  // if(sLock.IsLocked())  // {  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // sLock.Unlock();  // }  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  //  //UINT ThreadProc2(LPVOID lpParam)  //{  //  // CSingleLock sLock(gpmutex);//用互斥量对象创建一把锁  // for(int i=1;i<=100000000;i++)  // {  // //加锁，这种互斥锁有可能没有锁住，需要做个判断  // sLock.Lock();  // if(sLock.IsLocked())  // {  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // sLock.Unlock();  // }  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  ////使用信号量的方式  //UINT ThreadProc1(LPVOID lpParam)  //{  // CSingleLock sLock(gpsmph);//用互斥量对象创建一把锁  // for(int i=1;i<=100000000;i++)  // {  // //加锁，这种互斥锁有可能没有锁住，需要做个判断  // sLock.Lock();  // if(sLock.IsLocked())  // {  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // sLock.Unlock();  // }  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  //  //UINT ThreadProc2(LPVOID lpParam)  //{  //  // CSingleLock sLock(gpsmph);//用互斥量对象创建一把锁  // for(int i=1;i<=100000000;i++)  // {  // //加锁，这种互斥锁有可能没有锁住，需要做个判断  // sLock.Lock();  // if(sLock.IsLocked())  // {  // k \*=2;  // k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  // total += k;  // //解锁  // sLock.Unlock();  // }  // }  // ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  // return 0;  //}  //使用事件对象的方式  UINT ThreadProc1(LPVOID lpParam)  {  CSingleLock sLock(gpevent);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  //使用事件对象的方式，解锁后需要设置事件，也就是发信号说资源可以用了  gpevent->SetEvent();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  UINT ThreadProc2(LPVOID lpParam)  {    CSingleLock sLock(gpevent);//用互斥量对象创建一把锁  for(int i=1;i<=100000000;i++)  {  //加锁，这种互斥锁有可能没有锁住，需要做个判断  sLock.Lock();  if(sLock.IsLocked())  {  k \*=2;  k /=2;//这两句代码没有什么实际作用，只是用来模拟有很多业务逻辑需要完成  total += k;  //解锁  sLock.Unlock();  //使用事件对象的方式，解锁后需要设置事件，也就是发信号说资源可以用了  gpevent->SetEvent();  }  }  ::SetDlgItemInt(AfxGetApp()->m\_pMainWnd->m\_hWnd,IDC\_RESULT,total,FALSE);  return 0;  }  // 用于应用程序“关于”菜单项的 CAboutDlg 对话框  class CAboutDlg : public CDialogEx  {  public:  CAboutDlg();  // 对话框数据  enum { IDD = IDD\_ABOUTBOX };  protected:  virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持  // 实现  protected:  DECLARE\_MESSAGE\_MAP()  };  CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)  {  }  void CAboutDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)  END\_MESSAGE\_MAP()  // CLesson56threadsychronizeDlg 对话框  //构造函数  CLesson56threadsychronizeDlg::CLesson56threadsychronizeDlg(CWnd\* pParent /\*=NULL\*/)  : CDialogEx(CLesson56threadsychronizeDlg::IDD, pParent)  {  m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);  //创建临界区对象并且用全局指针变量结收  gpcs = new CCriticalSection();  gpmutex = new CMutex();  gpsmph = new CSemaphore(1,1);  gpevent = new CEvent(TRUE);//TRUE表示初始的时候是有信号的。  }  //析构函数  CLesson56threadsychronizeDlg::~CLesson56threadsychronizeDlg()  {  delete gpcs; //删除对象  gpcs = NULL; //把指针赋值为NULL  delete gpmutex;  gpmutex = NULL;  delete gpsmph;  gpsmph = NULL;  delete gpevent;  gpevent = NULL;  }  void CLesson56threadsychronizeDlg::DoDataExchange(CDataExchange\* pDX)  {  CDialogEx::DoDataExchange(pDX);  }  BEGIN\_MESSAGE\_MAP(CLesson56threadsychronizeDlg, CDialogEx)  ON\_WM\_SYSCOMMAND()  ON\_WM\_PAINT()  ON\_WM\_QUERYDRAGICON()    ON\_BN\_CLICKED(IDC\_BUTTON1, &CLesson56threadsychronizeDlg::OnBnClickedButton1)  ON\_BN\_CLICKED(IDC\_BUTTON2, &CLesson56threadsychronizeDlg::OnBnClickedButton2)  END\_MESSAGE\_MAP()  // CLesson56threadsychronizeDlg 消息处理程序  BOOL CLesson56threadsychronizeDlg::OnInitDialog()  {  CDialogEx::OnInitDialog();  // 将“关于...”菜单项添加到系统菜单中。  // IDM\_ABOUTBOX 必须在系统命令范围内。  ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);  ASSERT(IDM\_ABOUTBOX < 0xF000);  CMenu\* pSysMenu = GetSystemMenu(FALSE);  if (pSysMenu != NULL)  {  BOOL bNameValid;  CString strAboutMenu;  bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);  ASSERT(bNameValid);  if (!strAboutMenu.IsEmpty())  {  pSysMenu->AppendMenu(MF\_SEPARATOR);  pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);  }  }  // 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动  // 执行此操作  SetIcon(m\_hIcon, TRUE); // 设置大图标  SetIcon(m\_hIcon, FALSE); // 设置小图标  // TODO: 在此添加额外的初始化代码  return TRUE; // 除非将焦点设置到控件，否则返回 TRUE  }  void CLesson56threadsychronizeDlg::OnSysCommand(UINT nID, LPARAM lParam)  {  if ((nID & 0xFFF0) == IDM\_ABOUTBOX)  {  CAboutDlg dlgAbout;  dlgAbout.DoModal();  }  else  {  CDialogEx::OnSysCommand(nID, lParam);  }  }  // 如果向对话框添加最小化按钮，则需要下面的代码  // 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，  // 这将由框架自动完成。  void CLesson56threadsychronizeDlg::OnPaint()  {  if (IsIconic())  {  CPaintDC dc(this); // 用于绘制的设备上下文  SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);  // 使图标在工作区矩形中居中  int cxIcon = GetSystemMetrics(SM\_CXICON);  int cyIcon = GetSystemMetrics(SM\_CYICON);  CRect rect;  GetClientRect(&rect);  int x = (rect.Width() - cxIcon + 1) / 2;  int y = (rect.Height() - cyIcon + 1) / 2;  // 绘制图标  dc.DrawIcon(x, y, m\_hIcon);  }  else  {  CDialogEx::OnPaint();  }  }  //当用户拖动最小化窗口时系统调用此函数取得光标  //显示。  HCURSOR CLesson56threadsychronizeDlg::OnQueryDragIcon()  {  return static\_cast<HCURSOR>(m\_hIcon);  }  //定义防止按下车就退出程序的函数  BOOL CLesson56threadsychronizeDlg::PreTranslateMessage(MSG\* pMsg)  {  if (pMsg->message == WM\_KEYDOWN)  {  switch (pMsg->wParam)  {  case VK\_RETURN: //屏蔽回车键  return TRUE;  case VK\_ESCAPE: //屏蔽ESC键  return TRUE;    default:  break;  }  }  return CDialog::PreTranslateMessage(pMsg);  }  void CLesson56threadsychronizeDlg::OnBnClickedButton1()  {  // TODO: 在此添加控件通知处理程序代码  AfxBeginThread(ThreadProc1,NULL);  }  void CLesson56threadsychronizeDlg::OnBnClickedButton2()  {  // TODO: 在此添加控件通知处理程序代码  AfxBeginThread(ThreadProc2,NULL);  } |