基于 VC 的 Win32 多线程同步问题

The Multithread Synchronization in WIN32 Based On VC

王日宏（青岛建筑工程学院计算机系 266033）

摘要：介绍了线程的概念及 Win32 下线程同步的几种方法，给出了用 VC 下 MFC 类库实现读者写者问题的实例。

关键词：线程 多线程 同步 MFC 类库

1 引言

随着计算机技术的发展和应用的深入，自 80 年代中期提出线程概念至今，可以说线程技术已经相当成熟了，采用多线程的程序设计现已被广泛采用在操作系统、数据库管理系统和应用软件中。基于线程的多任务系统允许一个程序的两个或多个部分同时执行，这就增加了程序的维数，程序员可以通过定义分离的执行线路来管理程序的执行，使的应用程序更加高效，在许多电子测量处理系统中，采用多线程技术可以使采集与处理分开进行，这样可极大地提高系统效率。同时由于每个应用程序都在自己的线性空间中运行，因此系统的稳定性也会得以提高。本文着重说明了解决线程同步的几个方法，并给出了用 VC＋＋6.0 中的 MFC 类库实现的多线程编程实例——读者写者问题。

2 多线程的几个基本问题

2.1 线程概念

Windows98 /2K 和 NT 都是多任务操作系统，它们支持两种类型的多任务：基于进程（process）的多任务和基于线程（thread）的多任务。进程可认为是应用程序的一次执行过程，多进程就是系统可以同时执行多个任务。线程是进程中的一个实体，是被系统独立调度和分派的基本单位。一个进程至少有一个线程，即主线程，也可以有多个线程协同工作。进程从主线程开始执行，进而可以创建一个或多个附加线程来执行该进程内的并发任务，这就是基于线程的多任务。在 Win32 系统中，进程在独立的进程空间上运行，可以使用多达 4GB 的线性地址空间。但实际上进程本身是一个静态的概念，并不执行代码，程序的执行是由线程完成的。一个进程内的所有线程使用相同的 32 位线性地址空间，并共享所有的进程资源（包括打开的文件和动态分配的内存），但每个线程有自己的堆栈和 CPU 寄存器，称为线程的上下文（CONTEXT），随 CPU 型号不同而异，线程的执行由系统调度程序控制。每个线程都有相应的优先级，系统根据线程的优先级来调度它们。优先级共 32 级（0－31），称为基本优先级别（base priority level）。0 为最低优先级，31 为最高。0－15 级是普通优先级，16－31 级是实时优先级。线程的实际优先级设置是其所属的进程的优先级加上线程本身的优先级别。即：线程优先级 = 进程优先级 + 线程相对优先级线程的相对优先级有从进程优先级＋2 到－2 五档。此外还有两个较特殊的线程优先级：HREAD_PRIORITY_IDLE 其基本优先级对于实时优先级进程为 16，其余为 1；HREAD_PRIORITY_TIME_CRITICAL 基本优先级对于实时优先级进程是 31，其余为 15。

2.2 线程间的通信与同步

我们知道只有满足 Bernstein 条件的任务才能并发执行，否则程序将失去可再现性，甚至引发灾难性的后果。所以使用多进程与多线程时，需要协同两种或多种动作，此过程就称同步（Synchronization）。引入同步机制的一个原因是为了控制线程之间的资源同步访问，即互斥使用临界资源，另一个原因是协调线程之间的动作，让他们以指定的次序发生。

通常线程用于完成主程序中的某种任务，这就要求在主程序（可看成是线程）和线程之间有一个通信通道。为此可使用全局变量，消息及事件对象。使用全局变量在线程之间通信是一种很初级的方法，如果不清楚 c＋＋在汇编语言级上如何处理变量，那么使用全局变量将是很危险的。线程向主程序通信的最简单方法是在程序中加入用户定义的 windows 消息，定义消息之后可从线程中调用函数：：PostMessage()或 CwinThread：：PostThreadMessage()向线程发送消息。此外还可用事件对象，线程可以一直等待事件成为有信号，然后在适当的时候完成通信操作。事件对象在 MFC 中由 Cevent 类描述，创建 Cevent 对象后，事件处于无信号状态。可调用事件的 SetEvent()成员函数使事件处于有信号状态，用于完成等待事件信号的函数为 WaitForSignleObject()。

对于多线程程序中同步问题，常用的解决方法有四种同步对象：临界区（Critical Section）、互斥量（Mutex Semaphore）、信号量（Semaphore）和事件对象（Event）。临界区对象通过提供一个进程内所有线程必须共享的对象来控制线程。互斥量的工作方式很类似于临界区，只是互斥量不仅

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

保护一个进程内为多个线程使用的共享资源，而且还可以保护系统中多个进程之间的的共享资源。信号量可以允许一个或有限个线程访问共享资源。它是通过计数器来实现的，初始化时赋予计数器以可用资源数，当将信号量提供给一个线程时，计数器的值减1，当一个线程释放它时，计数器值加1。当计数器值小于等于0时，相应线程必须等待。从本质上讲，互斥量是信号量的一种特殊形式。事件对象作为标志在线程间传递信号。同步问题是多线程编程中最复杂的问题，下面通过读者写者问题说明多线程同步的实现方法。

3 基于VC++6.0多线程同步的实现

用VC++实现多线程的编程有多种，一种是利用Win32 API函数编写C风格的Win32应用程序，另一种是利用MFC类库编写C++风格的应用程序。两种方法各有优缺点，基于Win32 API的应用程序代码小巧，运行效率高，但程序员需要编写较多的代码；基于MFC的应用程序开发速度较快，但MFC类库代码很大，应用程序的执行代码离不开它们。用AfxBeginThread()或new CwinThread类产生的线程能接受MFC消息，而用run_time函数_beginthreadex创建的线程不能使用任何MFC API，下面以MFC为例。

3.1 读者－写者问题

一个数据对象可以被多个线程共享，有些线程要求读，有些线程要求对数据对象进行写或修改。这种共享的数据对象是不允许一个写线程和其它读线程，或者两个以上的写线程同时访问的。读者－写者问题就是指保证一个写线程必须与其它线程互斥地访问共享对象的同步问题，它是一个经典的进程同步问题。我们以读者优先的读者－写者问题为例，说明进程间的同步。为简化说明同步问题，假设有5个读者线程，2个写者线程。建立一个基于MFC的对话框程序，用一个开始按钮创建线程，用一个退出按钮结束程序，每个读者或写者都对应一个Edit Box，用以显示它们当前的状态。读者、写者在不断地进行读写操作，直至结束程序。申请读写时刻及读写时间由产生的随机数确定。下面给出相关的线程及创建线程的代码。

3.2 线程函数的实现文件

说明部分：

```
#define READERNUM5 //读者数
#define WRITERNUM2 //写者数
CCriticalSection wsema; //用以互斥使用共享数据对象资源
CMutex rsema; //用以互斥访问"读者数量"资源
BOOLEAN bDone = false; //线程运行标记，在退出按钮处理函数中置为true，结束线程；
int greadcount=0; //标记正在读的读者数量
struct ThreadParam { CEdit* pEdit; //相应读者写者对应的edit box
};
```

线程实现部分：

```
UINT WriterThread(LPVOID pParam) //写者线程
{ CEdit* pEdit = ((ThreadParam *)pParam)->pEdit;
    while (! bDone) {pEdit -> SetWindowText(CString("准备进入!"));
        wsema.Lock(); //申请使用临界资源
        pEdit->SetWindowText(CString("正在写!"));
        Sleep(GetRandomTime());
        wsema.Unlock(); //释放资源
        pEdit -> SetWindowText(CString("离开!"));
        Sleep(GetRandomTime());}
  return 0;}
UINT ReaderThread(LPVOID pParam) //读者线程
{ CEdit* pEdit = ((ThreadParam *)pParam)->pEdit;
  while (! bDone) {Sleep(GetRandomTime());
        pEdit->SetWindowText(CString("申请读互斥"));
        rsema.Lock();
        if (greadcount==0){pEdit->SetWindowText(CString("申请写互斥"));
                wsema.Lock();}
    greadcount=greadcount+1;
        rsema.Unlock();
        pEdit->SetWindowText(CString("正在读!")); //Perform read operation
        Sleep(GetRandomTime());
        rsema.Lock();
        greadcount=greadcount-1;
        if(greadcount==0)wsema.Unlock();
        rsema.Unlock();
        pEdit->SetWindowText(CString("已离开!")); }
  return 0;}
```

创建线程部分：

```
void CReadwriterDlg::OnStart()
{CEdit * pReadEditArray[READERNUM] = {&m_ed-
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

```
it1, &m_edit2, &m_edit3,&m_edit4, &m_edit5};
CEdit * pWriteEditArray[WRITERNUM] = {&m_edit6, &m_edit7};
GetDlgItem ( IDC _ START ) - > EnableWindow (FALSE);
int k;
for (k = 0; k < READERNUM; k++ ) { //建读者线程
  ThreadParam * readparam = new ThreadParam;
  readparam->pEdit = pReadEditArray[k];
  AfxBeginThread ( ReaderThread, ( LPVOID) readparam,THREAD_PRIORITY_NORMAL,0,0,NULL); }
for (k = 0; k < WRITERNUM; k++ ) { //建写者线程,参数含义同读者线程
  ThreadParam * writeparam = new ThreadParam;
  writeparam->pEdit = pWriteEditArray[k];
  AfxBeginThread ( WriterThread, ( LPVOID ) writeparam,THREAD_PRIORITY_BELOW_NORMAL,0,0,NULL);}}
```

产生随机时间函数为：

```
int GetRandomTime( ){ return (rand()%4000 + 500);}
```

3.3 关于 MFC 多线程编程

MFC 支持工作者线程和用户界面线程两种类型的线程。工作者线程没有消息机制，常用来实现象计算、后台打印这样的任务，不需要与用户交互；MFC 为用户界面线程提供消息机制，用来处理用户输入。这两种线程类都是从 CWinThread 类派生而来的，不同的是，对于工作者线程不需要显式的创建 CWinThread 对象，而由 AfxBeginThread()自动创建。在 MFC 应用程序中，主线程是从 CWinApp 类派生来的。事实上，CWinApp 就是用户界面线程的一个典型实例，它是从 CWinThread 派生而来。在 MFC 中，有如下派生关系：CObject->CCmdTarget->CWinThread->CWinApp。

MFC 封装了线程同步操作，提供了一组同步类和同步访问类来解决同步问题。同步类有：CSyncObject，CSemaphore，CMutex，CCriticalSection，CEvent。CSyncObject 是其余四个类的基类，不直接使用，两个同步访问类是 CSingleLock 和 CMultiLock。同步访问类要和同步对象配合使用。使用同步类时程序中要包含头文件"afxmt.h"。MFC 四种线程同步类的使用方法基本上是一致的，很多情况下不需要同步访问类，直接调用它们自身的成员函数 Lock()和 Unlock()即可。

在上例中，我们用了 Cmutex 和 CcriticalSection，用 CcriticalSection 作为读者与写者之间的对临界资源使用的互斥，用 Cmutex 作为读者间访问读者计数器的互斥量。其实理论上对这一问题中的两个互斥量，用信号量、互斥量和临界区哪两个同步类都是可以的，但在 WIN98 下若用两个 Cmutex 同步类对象作为读者和写者的互斥量，应用程序不是很稳定，如改用 CSemaphore 同步类对象，则系统运行稳定、安全，与前面所述"信号量是 Win98 /NT 同步系统的核心"一致。所以同一进程内的各线程同步可选用信号量、临界区或事件同步类即可，不必用实现稍复杂的互斥同步类，当考虑不同应用程序共享资源时再用互斥同步类。因为我们引入多线程的目的在于能使系统更高效、安全地运行。

4 结束语

多线程技术适于需要并发执行的应用程序，尤其对低速外设的操作或者大部分时间被阻塞的任务，可以设一个独立的线程来完成，这可提高 CPU 的利用率，但多线程编程是一门比较复杂的技术，它不仅要求有熟练的编程技术，还要求对操作系统内部有深入的了解，尤其是线程间的同步问题。如何恰当的运用线程同步对象是编程中要着重处理的问题。多线程编程对于多处理器平台有很重要的意义，但对于单处理器平台，由于线程间的切换需要消耗一定的 CPU 资源，所以有时可能反而会降低系统的效率，设计时应加以注意。

参考文献

1 官章全、唐晓卫，Visual C++6.0 编程实例详解，电子工业出版社，1999,11。

2 William Stalling. Operating System Internals and Design Principles. Prentice - Hall International, Inc. 1998。

3 刘晓晶，WIN32 下的多线程编程，电脑编程技巧与维护，2002(8)：43-45。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn