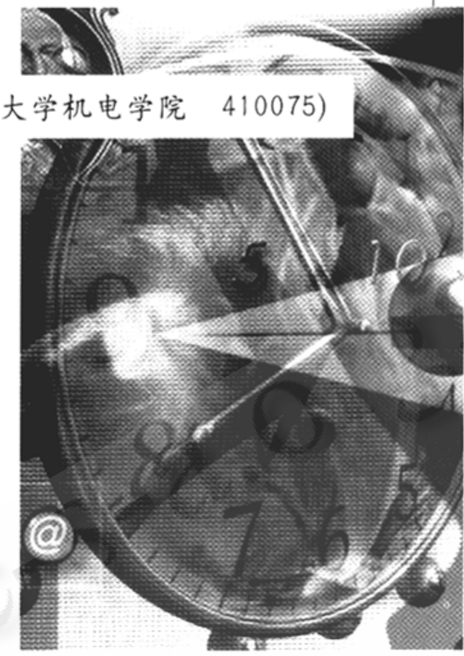
VC++应用程序精确定时方法的实现

姚 晔 胡益雄 （湖南长沙中南大学机电学院 410075）

摘要：本文对VC++普通定时器资源在开发工业应用程序中所存在的问题进行了剖析，同时提出了两种适合于实际需要的VC++应用程序精确定时方法，并对它们进行了详细得说明和比较。

关键词：VC++ 普通定时器 软件校正 多媒体定时器



1 前言

在许多工业生产测控系统中都需要使用系统提供的定时器，由于普通定时器是建立在IBM PC机硬件和ROM BIOS中系统定时的简单扩充，系统每秒钟只能产生18个时钟信号[2]，所以，即便其最小分辨精度能达到毫秒级，定时间隔的实际精度也不会超过1/18秒（约55毫秒），也就是每次产生定时器事件间隔的最大误差可达到55毫秒，如果将之累积，则随时间的推移，其产生的误差将会急剧增大。经实验发现如设定一个以200毫秒为时间间隔的普通定时器，每产生5次定时器事件，将秒计数单元加1作为累计的时间，同时，读取系统时间，即实际时间，则程序运行15分钟后，此累计时间比实际时间慢大约1秒钟，若定时器事件要完成的任务较多时，则二者的差值将更大。由此看来，在用VC++开发大型的工业应用程序的过程中，普通定时器的使用可能会导致检测数据的丢抄和控制的严重失调。为了使VC++应用程序能更好得满足工业生产的需要，很有必要对其定时方法进行探讨和研究，以保证足够的定时精度。

2 VC++精确定时方法

2.1 软件校正法

该方法利用软件实时检查系统时钟，并以此为依据，对普通定时器的定时间隔进行校正，从而得到比较精确的定时间隔。下面以产生1秒的精确时间为例对这种精确定时方法进行详细的介绍。

由实验表明，普通定时器5次200毫秒的定时比实际1秒的时间要长。这样，可将200毫秒作为普通定时器定时间隔的上限，而其下限则要根据定时器事件的任务多少来确定，如任务多，下限值就低，反之，则高。在本文实例中，用for(int i=0; i<10000000;i++)来模拟定时器事件的任务，经笔者反复调试之后，将100毫秒作为普通定时器定时间隔的下限值。在启动普通定时器的同时（普通定时器的初始定时间隔设定为上限值200毫秒），读取系统时钟作为起始值，而在每产生5次定时器事件后读一次系统时钟作为当前值，用当前值减去起始值得到实际计数值；同时，累加计数单元加1作为普通定时器的计数值。若普通定时器的计数值小于实际计数值，则表明普通定时器慢了，这时需要重新启动普通定时器，将其定时间隔设定为下限值100毫秒；反之，则将定时间隔重置为上限值200毫秒。

以VC++6.0为开发工具，具体的编程过程如下：

(1) 创建一个单文档应用程序工程，命名为“普通定时器校正”；

(2) 在"普通定时器校正View.h"头文件中，加入以下代码：

```
public:
    long account; // 用来表示修正后的普通定时器的计数值
    time_t m_time;// 用来记录计算机系统时间
```

同时，并在视的构造函数中将accuont进行初始化为0；

(3) 在"普通定时器校正View.cpp"中的OnInitialUpdate函数中启动一个定时间隔为200毫秒的普通定时器，并记录计算机系统当前时刻。程序代码为：

```
void CMy1View::OnInitial-Update()
  {
      CView::OnInitialUpdate();
    // 记录启动普通定时器的计算机当前时刻
```

```
CTime t=CTime::GetCurrentTime();
m_time=t.GetTime();
SetTimer(1,200,NULL);/// 设定一个普通定时器,定时间隔为200毫秒
```

(4) 使用ClassWizard添加WM_TIMER消息处理程序函数，在OnTimer函数中加入自己要处理的事件，并每隔1秒钟对普通定时器的定时间隔进行校正。程序代码如下：

```
void CMyView::OnTimer(UINT nIDEvent)
{
    static int count=0;
    count=count+1;
//如果count=5,则表明修正周期已到，普通定时器的定时值需要进行修正
    if(count==5)
{
    count=0;//将count重新置0,以便进行下一个周期的修正
    for(int n=0;n<10000000;n++);//模拟在实际应用工程中要处理的数据量
    static  short i=0;
    i++;
    account1+=1;// 普通定时器的计数值
// 每秒钟画一条直线，起点不变，终点定时变化
    CClientDC dc(this);
    CPen pen(PS_SOLID,5,RGB(0,255,0));
    CPen *pOldPen=dc.SelectObject(&pen);
    dc.MoveTo(400,100);
    dc.LineTo(480,180+i);
    dc.SelectObject(pOldPen);
// 显示普通定时器输出的计数值
    char temp [20];
    itoa(account1,temp,10);
    dc.TextOut(720,100,temp);
    dc.SetTextColor(RGB(0,255,0));// 设置文本的颜色
    dc.TextOut(400,100,"修正后的普通定时器定时输出的计数值:");// 文字说明
//从普通定时器开始启动后的实际计数值（以秒为单位）
    char temp1 [20];
    CTime t=CTime::GetCurrentTime();// 获得计算机系统当前时间
    time_t interval=t.GetTime()-m_time;// 得到实际计数值
// 显示实际计数值
    ltoa(interval,temp1,10);
    dc.SetTextColor(RGB(0,0,0));// 设置文本的颜色
    dc.TextOut(500,50,temp1);
    dc.TextOut(390,50," 实际计数值：");// 文字说明
//修正后的普通定时器定时（每隔一秒钟）输出的计数值与实际计数 // 值的差值比较说明
    char temp2 [20];
    ltoa(fabs(interval-account1),temp2,10);
    dc.TextOut(150,500," 修正后的普通定时器输出的计数值与实际计数值差值：");
    dc.TextOut(590,500,temp2);
//对普通定时器的定时间隔进行调整修正
//如果普通定时器输出的计数值比实际计数值小，则进行以下修正
    if(account1<interval)
{
    KillTimer(1);//释放定时器
    SetTimer(1,100,NULL);//重新启动普通定时器，定时间隔为100毫秒
}
// 否则进行以下修正
    else
{
    KillTimer(1); //释放定时器
    SetTimer(1,200,NULL); // 重新启动普通定时器，定时间隔为100毫秒
}
}
CView::OnTimer(nIDEvent);
}
```

(5) 由于在OnTimer函数中用到了fabs(x)函数，所以还要在"普通定时器校正 View.cpp" 前面添加代码：#include "math.h"。

(6) 编译运行，可以看到不管程序运行多久，校正后的普通定时器的计数值与实际计数值之间的差值不会超过1。

2.2 使用多媒体定时器

Microsoft公司在其多媒体Windows中提供了高精度定时器的低层API支持，使应用程序可以得到周期性的时间中断，无论应用程序在做什么工作，操作系统都能在多媒体定时器事件到来时打断该程序，而先去调用多媒体定时器的回调函数。利用多媒体定时器可以很精确地读出系统的当前时间，并且能在非常精确的时间间隔内完成一个事件、函数或过程的调用。在VC++环境下定制和使用多媒体定时器的步骤如下：

(1) 设定多媒体定时器的相关参数。设定多媒体定时器相关参数的主要函数有：TimeGetDevCaps函数、TimeBeginPeriod函数和TimeEndPeriod函数。

TimeGetDevCaps函数用来确定定时器服务提供的最大和最小事件

周期，该参数可以得到用户想要设置的事件触发周期。TimeBeginPeriod函数和TimeEndPeriod函数可以为系统设置和消除最小时钟事件精度。在具体的应用程序中，应根据需要设置定时器精度。

(2) 启动多媒体定时器事件。可用TimeSetEvent初始化和启动时间事件，同时给出定时器回调函数的入口地址。其函数原型为：

```
MMRESULT timeSetEvent(UINT uDelay, UINT uResolution, LPTIMECALLBACK LpFunction, DWORD dwUser, UINT fuEvent);
```

其中的参数说明：

uDelay 延迟时间（单位为毫秒）；

uResolution 时间精度（单位为毫秒），其值越小，精度越高；在Windows中默认值为1毫秒；

LpFunction 定时器触发的事件的回调函数的地址；

dwUse 用户自定义的返回值；

fuEvent 定时类型，可取TIME_ONESHOT、TIME_PERIODIC等值，其中，TIME_PERIODIC表示事件每uDelay毫秒发生一次。

(3) 回调函数的编写。LpFunction表示事件回调函数的入口地址，但具体的回调函数则需要编程人员自己编写。在这里需要注意的是，该函数是Windows API类型的函数，不能将它定义为某个具体类的成员，当需要调用MFC类中某个具体的成员函数时，则需要先获取这个类的指针，然后才能引用该类的成员。

(4) 删除定时器以释放系统资源。定时器是一种有限的资源，不能无休止地创建，在使用完后要及时将它删除掉。可用函数timeKillEvent来完成这项工作，其原型为：

```
MMRESULT timeKillEvent (UINT uTimeID);
```

其中，uTimeID为由timeSetEvent返回的定时器标识号。

下面，笔者提供一个编程实例来进一步说明多媒体定时器在VC++6.0环境下的具体用法。

① 同样创建一个单文档的应用工程，取名为“多媒体定时器”；

② 在“多媒体定时器View.h”中加入自己定义的函数声明：

```
public:
    LRESULT OnMessage (WPARAM wParam, LPARAM lParam);
```

在这个函数中，我们可以完成想要作的事情；同时，在protected:处加入一个计数变量定义：long account; 用来表示多媒体定时器的计数值。并在视的构造函数中进行初始化为0；

在“多媒体定时器View.cpp”中的“# endif”后面加入如下声明：

```
UINT timerID; //定时器的标识号
UINT wTimerRes;
void CALLBACK LpFunction ( UINT timeID, UINT msg, DWORD dwUser,
    DWORD dwUser, DWORD dw1, DWORD dw2);
```

③在“多媒体定时器 View.cpp”中的OnInitialUpdate函数中加入参数设置函数和多媒体定时器启动函数：

```
timeBeginPeriod(wTimerRes);
timeEndPeriod(wTimerRes);
timerID=timeSetEvent(1000, 10, LpFunction, 30, TIME_PERIODIC);
```

其中，定时器每隔一秒钟启动一次事件，事件的回调函数地址为LpFunction;

④在“多媒体定时器 View.cpp”中编写自定义函数OnMessage和回调函数LpFunction：

```
LRESULT CMyView::OnMessage(WPARAM wParam, LPARAM lParam)
{
    for(int n=0;n<10000000;n++);//模拟在实际应用工程中要处理的数据量
    static   short i=0;
    i++;
    account+=1;// 每隔一秒多媒体记数值加1
    // 每秒钟画一条直线，起点不变，终点定时变化
    CClientDC dc(this);
    CPen pen(PS_SOLID,5,RGB(255,0,0));
    CPen *pOldPen=dc.SelectObject(&pen);
    dc.MoveTo(100,100);
    dc.LineTo(180,180+i);
    dc.SelectObject(pOldPen);
    // 定时输出计数值
    char temp [20];
    itoa(account,temp,10);
    dc.SetTextColor(RGB(0,0,0));// 设置计数值字符的颜色
    dc.TextOut(350,100,temp);
    dc.SetTextColor(RGB(255,0,0));// 设置文本的颜色
    dc.TextOut(100,100," 多媒体定时器定时输出的计数值：");
    return 0;
}
void CALLBACK LpFunction( UINT timerID, UINT msg,
    DWORD dwUser, DWORD DWORD dw2)
{
    //获得应用程序指针和第一个文
```

档模板的位置

```
POSITION pos=AfxGetApp()->GetFirstDocTemplatePosition();
//获得第一个文档模板的指针
CDocTemplate *pDocTemplate=AfxGetApp()->GetNextDocTemplate(pos);
//获得与文档相关联的文档列表中
pos=pDocTemplate->GetFirstDocPosition();
//获得与文档相关联的文档指针
CDocument *pDoc=pDocTemplate->GetNextDoc(pos);
//获得与文档相关联的视的位置
pos=pDoc->GetFirstViewPosition();
//获得与文档相关联的视的指针
CMyView *pView=(CMyView*)pDoc->GetNextView(pos);
//调用视的成员函数，即任务执行函数
pView->OnMessage(8,9);
}
```

⑤ 在“多媒体定时器View.cpp”中加入多媒体必须的头文件

```
#include "mmsystem.h"
```

⑥ 在Project菜单中打开Settings对话框，并在Link项中加入程序连接需要的库文件 winmm.lib;

⑦ 编译运行即可看到程序运行结果。

3 结论

在用VC++进行开发工业测控应用软件的过程中，精确定时是保证软件性能优良的一个重要方面。本文所提出的两种精确定时方法，在笔者开发的“ET18型智能化列车空调性能测控系统”中得到了实际的应用，并收到了很好的效果。在运用当中应该注意以下几点：

(1) 多媒体定时器的定制和使用，一个关键问题是定时器事件回调函数的编写。在编写回调函数时，要注意此函数是Windows API类型的函数，，若想利用MFC类的成员函数完成需要处理的事件，必须利用VC++中提供的一些全局函数来获得MFC类的指针，然后方可使用其成员函数或成员变量。另外，为了减轻系统的负担，定时精度值应该选择适合于应用程序的最大值。

(2) 使用软件校正法进行精确定时，关键问题是定时器间隔时间下限值的确定。在实际过程中，应根据不同情况，并通过反复调试，然后再确定其最优的下限值。

(3) 定时精度在500毫秒以上时，本文提供的两种方法能很好得达到预期效果；但定时精度需要控制在500毫秒以下时，最好还是使用多媒体定时器进行定时操作。■

参考文献

1 .David J.Kruglingski著，王国印译， Visual C++4.0技术内幕，清华大学出版社，1998。

2 李宝琛、吴发文等， 微型计算机常用器件手册， 福建科学技术出版社，1985。

3 Leinecker R C， Visual C++开发技术内幕[M]， 北京机械工业出版社，1999。

4 木林森、高峰霞、罗丽琼等，Visual C++ 6.0使用指南与开发[M]，北京清华大学出版社，1998。