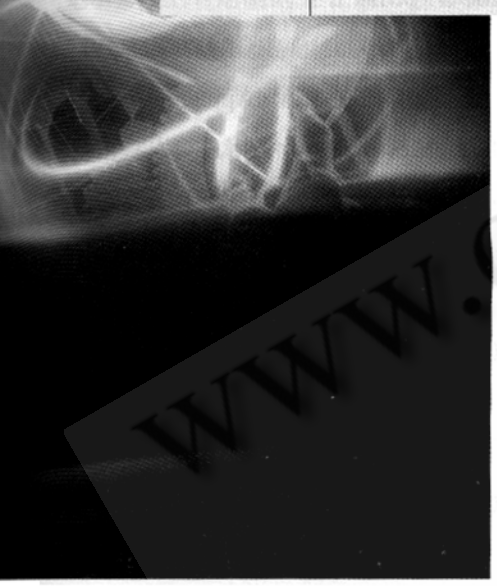
基于MFC编程的闲置处理优化控制

向卫军 (武汉中国舰船研究院第七一七研究所 430074)

摘 要 闲置处理过于频繁是基于AppWizard编写的MFC应用程序执行效率低、CPU占用率高的直接原因。本文介绍基于MFC编程闲置处理控制的优化方法。包括重载CWinThread::IsIdleMessage ()函数避免闲置处理过于频繁，提高基于MFC应用程序的执行效率、降低CPU占用率；利用Visual C++中未公开的WM_KICKIDLE消息控制系统直接执行闲置处理、提高界面刷新的实时性。

关键词 MFC 消息循环 闲置处理 重载 继承



1 概述

Visual C++ 6.0是Microsoft至今最全面和最完善的程序开发产品之一，它所包含的MFC(Microsoft Foundation Class)库封装了大量的标准类，是创建Windows应用程序非常优秀的类库。

微软的系统程序员为使用Visual C++开发多种应用程序设计好了程序的框架部分，通过AppWizard就可以自动生成包含了一些基本功能的应用程序框架，开发者在此基础上进行程序的设计，编程难度比基于API编程大大降低了，是Windows应用程序走向普及的重要因素之一。

虽然利用MFC AppWizard编写的Windows应用程序具有标准的框架结构、优秀的可移植性和可继承性；但是执行效果却差强人意。主要表现为CPU占用率高，不适合编写实时性要求较高的控制软件。限制了MFC编程的适用范围。

2 消息循环和闲置处理介绍

Windows操作系统是基于事件驱动的交互式系统，它监测事件发生并将包括用户输入在内的各种事件以消息的形式发送到应用程序的消息队列。MFC应用程序通过CwinApp类的成员函数Run( )接收并处理Windows发来的消息。简而言之，消息循环就是函数Run( )中的一段循环执行的程序，它监测并从消息队列里提取待处理的消息发送到相应的处理部分执行。

消息被应用程序分配执行后，消息循环调用CWinThread::IsIdleMessage () 函数，判断消息队列中是否有待处理的消息。如果没有消息该函数返回TRUE，消息循环就执行闲置处理函数，处理更新工具栏按钮、状态指示器等工作，并完成在运行过程中创建的临时对象的清理工作。研制处理是Windows程序必不可少的组成部分之一，通过重载这个函数还可以完成某些自定义的后台处理任务。

消息循环分发消息的速度很快，大部分时间消息队列都是空的，所以周期性消息（例如鼠标移动、定时器事件等）会导致闲置处理被频繁执行。闲置处理过于频繁会消耗较多的CPU时间，甚至导致消息堆积，降低系统的反应实时性，是基于AppWizard编写的MFC应用程序执行效率低、CPU占用率高的直接原因。

闲置处理函数原型为：virtual BOOL CWinApp::OnIdle(LONG lCount)，它的执行方式如下：

(1) 如果消息循环检测消息队列发现没有待处理的消息，它就调用OnIdle处理程序，同时对参数lCount置0。

(2) OnIdle完成一些处理后返回，如果返回一个非零值就表明OnIdle将被再次调用并且执行进一步的处理。

(3) 消息循环再次检测消息队列。如果没有待处理的消息，就再次调用OnIdle并增加lCount参数值。

(4) 直到最后，OnIdle完成了所有的闲置任务并且返回0。从而告诉消

息循环停止调用OnIdle直到消息队列接收到下一个消息。

消息循环直到闲置处理返回以后才能恢复消息循环、响应用户输入。为了避免阻塞消息循环，提高用户输入的响应速度，不能重载OnIdle函数执行过长的用户自定义任务。

周期性事件在应用程序中经常出现，例如定时器事件、周期性发送消息或鼠标移动等，导致闲置处理频繁执行。即便是基类成员函数CWinApp::OnIdle，除了处理用户界面更新外（例如菜单条、工具条按钮刷新等），还要执行内部临时数据结构清除等操作。闲置处理频繁执行状况下的时间开销是不容忽视的，并且后面的实验数据也证明了周期性事件导致频繁执行。

闲置处理提高了CPU占用率，使整个进程的反应实时性打折扣。所以对基于MFC编程的闲置处理实现自定义控制是提高应用程序性能的重要手段之一。

3 控制闲置处理有选择执行

事实上，MFC程序的闲置处理过于频繁，避免闲置处理函数不必要的执行，对提升系统的执行效率具有积极的作用。不幸的是，我们无法直接修改闲置处理的调用机制改变这种状况。因为消息循环结构和消息处理方式被封装在程序框架内部，直接修改难度大，且会导致软件的向下兼容性、可移植性变差。

MFC优秀的可继承性，使我们可以通过函数重载来实现定制闲置处理。

MFC应用程序实例是CwinApp类的对象，而CwinApp类的基类是CwinThread类。所以MFC应用程序可以重载CwinThread类的成员函数。消息循环是通过基类虚函数CWinThread::IsIdleMessage判断是否需要执行闲置处理的。

通过重载CWinThread::IsIdleMessage（）函数定制闲置处理调用，实现消息队列空时有选择的执行闲置处理。避免某些频繁的消息处理中，系统不加选择的执行闲置处理。从而降低CPU占用率，提高MFC编程中消息循环的执行效率，提高程序处理的实时性。编程实践证明，经以上方法优化处理后的程序可以达到一般工业环境下的实时控制要求。

消息循环把消息分配出去的时候，调用虚函数virtual BOOL IsIdleMessage(MSG* pMsg)检测消息队列。如果消息队列中没有其他待处理的消息。虚函数IsIdleMessage返回值为TRUE，当前消息pMsg被处理后将执行闲置处理；否则返回值为FALSE，不执行闲置处理。通过重载IsIdleMessage函数，检索消息pMsg，可以避免特定消息处理后执行闲置处理。

基于MFC编程时，函数重载通常在VC++的辅助工具ClassWizard中处理。ClassWizard是Visual C++中相当好用的辅助工具，功能包括用来管理可重载的函数。但是可以观察到ClassWizard中由CwinThread类派生的CmyApp类消息列表中并没有列出IsIdleMessage选项，所罗列出来的只是CwinApp类的成员函数，而IsIdleMessage是CwinApp类的基类CwinThread的成员函数，不被ClassWizard直接支持。

既然系统能支持对基类成员函数的重载，我们就可以用手工添加的方法实现函数重载。下面的示例程序可以演示定制闲置处理的结构和实现方法。程序具体实现鼠标移动、定时器消息和用户自定义消息处理后避免执行闲置处理，其他条件下当消息队列空闲时执行闲置处理。

首先，在头文件"CMyApp.h"中添加对函数原型的说明，其中黑体部分是需要手工加入的部分，其他部分是对黑体部分所处位置的标示：

```
//{{AFX_VIRTUAL(CMyApp)
    public:
    virtual BOOL InitInstance();
virtual BOOL IsIdleMessage(
MSG* pMsg );
    virtual int  ExitInstance();
```

然后，在"CMyApp.cpp"中重载IsIdleMessage函数，用户可以根据自己的需要对函数体部分作必要的修改。程序如下所示：

```
BOOL CMyApp::IsIdleMessage
(MSG* pMsg)
{
if (!CWinApp::IsIdleMessage
(pMsg))
    return FALSE; // 消息队列有
待处理的消息
    // 消息队列没有待处理的消
息……
  if ((pMsg->message >=
  WM_MOUSEFIRST &&
  pMsg->message <=
  WM_MOUSELAST) ||
    (pMsg->message >=
    WM_NCMOUSEMOVE &&
    pMsg->message <=
   WM_NCMBUTTONDBLCLK))
    return FALSE; // 鼠标移动不
进行闲置处理……
    if (pMsg->message ==
WM_TIMER)
    return FALSE; //定时器事件不
进行闲置处理……
    if (pMsg->message >=
```

```
WM_UserMin &&
        pMsg->message <=
WM_UserMax)
        return FALSE; // 自定义消息不进行闲置处理……
        return TRUE; // 除此之外，程序必需进入闲置处理……
    }
```

4 控制闲置处理直接执行

虽然很多时候调用闲置处理函数是不必要的，定制闲置处理可以降低CPU占用率和提高软件实时控制能力。但是上文的处理方法在特定条件下可能导致用户界面不能及时更新。

对于需要尽早执行闲置处理，来更新用户界面、对数据的变化做出响应的状况，可用一个简单直观的方法来解决。Visual C++开发软件包里有一个名为afxpriv.h的头文件，第124行定义了一个未公开的消息WM_KICKIDLE（#define WM_KICKIDLE 0x036A //causes idles to kick in ）。通过发送WM_KICKIDLE消息可以控制系统直接执行闲置处理，提高界面刷新的实时性。

通过如下步骤，可以控制程序立即执行闲置处理。

(1) 在头文件"stdafx.h"中添加如下代码：

```
#include <afxpriv.h>    // for WM_KICKIDLE
```

(2) 在需要执行闲置处理的位置添加如下代码：

```
PostMessage(WM_KICKIDLE);
```

消息循环执行到WM_KICKIDLE消息时，不论消息队列空闲与否，都直接执行闲置处理。

这种方法不影响程序的向下兼容性，阅读起来很直观。程序不执行多余的操作，直接执行闲置处理，具有执行效率高的优点。界面刷新不依赖于消息循环的状况，所以界面可以实时更新、迅速反应。

5 闲置处理优化数据对比和分析结语

为了验证上面的方法和获得简单的实验数据，编制了一个简单的基于MFC AppWizard的程序，除程序框架外，只包含一个频率可变的周期性WM_TIMER事件。测试重载IsIdleMessage(MSG* pMsg)函数条件下和缺省条件下的CPU占用率。表1是测试数据对比：第一组为缺省条件下的CPU占用率；第二组为重载IsIdleMessage(MSG* pMsg)函数，执行闲置处理条件下测试的CPU占用率。

从表1数据可以看到，控制闲置处理执行的次数对提高Windows应用程序消息循环的效率、降低CPU占用率能起到相当重要的作用。结合以上数据可以得出如下认识：

(1) 对于频率大于每秒20次的周期性消息，将会导致消息堆积。会导致周期性处理事件的处理实时性下降。以至于很难适合工业控制环境的需要。

(2) 频繁的周期性事件引发闲置处理，会消耗大量的CPU处理时间，导致软件反应迟钝，有明显地阻塞现象。直接导致基于这种编程方式下的应用程序不能满足实时性处理环境的起码要求。

(3) 即便是对于高速的应用系统，过多闲置处理对系统资源的浪费也是相当可观的。

(4) 通过重载虚函数CWinApp::IsIdleMessage(pMsg)，优化AppWizard闲置处理的缺省调用方式，对降低CPU占用率、提高程序的执行效率和反应实时性效果确切。

表 1 闲置处理优化效果对比

	计算机CPU占用率	
消息频率	优化前	优化后
5次/秒	8%	<1%
10次/秒	15%	<1%
20次/秒	31%	<1%
40次/秒	32%	<1%
100次/秒	32%	<1%

* 数据测试计算机主要配置为：PⅢ933CPU、256M内存、Win98操作系统

** CPU占用率测试软件为：Norton System Doctor.■

参考文献

1 *MSDN Library Visual Studio 6.0 版*

2 *Inside Visual C++ 6.0*