多线程与 ODBC 在串口编程中的应用

毕 博,李世光,高正中,徐坤山,刘隆吉
(山东科技大学 信息与电气工程学院,山东 青岛 266510)

摘 要:串口通信所要求的实时数据读取及储存占用了大量的系统资源。具有同时执行多任务特性的多线程技术和易操作的 ODBC 技术在 VC 下的综合利用很好地解决了这一问题。在水处理电压电流检测系统中利用此方法进行程序设计提高系统的实时性,取得了很好的设计效果。

关键词:多线程;ODBC;串口通信
中图分类号:TP311 **文献标识码**:B

Application of multithreading and ODBC in serial-port program

BI Bo, LI Shi Guang, GAO Zheng Zhong, XU Kun Shan, LIU Long Ji
(College of Information and Electrical Engineering, Shandong University of Science and Technology, Qingdao 266510, China)

Abstract: Real-time data reading and storage which are required in serial communication occupy a large amount of system resources. Utilizing multithreading technology which can implement multitasking at the same time and ODBC technology which can be easily operated in VC, has solved this problem well. Voltage and current of water treatment measuring system adopts this approach to improve system in real-time and obtains nicer design purpose.

Key words: multithreading; ODBC; serial communication

目前,串口通信技术广泛用于远程监控和工业自动化领域。计算机串口编程在通信软件中有着广泛的应用,如电话、传真、视频、控制和数据采集等。而在实际工程中各种实时数据的保存以及日后的查询功能往往是必不可少的。在 VC 中,利用多线程技术和 ODBC 技术能够十分方便有效地解决串口的实时通信以及数据储存、查询问题。

1 VC 中的多线程编程

1.1 多线程的建立与终止

VC++把线程分为两种:用户界面线程(UI Threads)和工作者线程(Worker Threads)。前者一般用于处理独立于其他线程执行之外的用户输入,即直接与用户进行人机交流;后者通常用来执行后台计算和维护任务,如冗长的计算过程,循环等待等。在串口通讯中,接受数据的操作需要时时查询,所以必须将循环等待数据的操作放入工作者线程中,让其在系统后台运行。

VC++中通过 AfxBeginThread()全局函数来创建一个线程,此函数在创建一个新线程的同时初始化了该线程。AfxBeginThread 的具体定义如下:

(1)CwinThread* AfxBeginThread (AFX_THREADPROC pfnThreadProc, LPVOID pParam, int nPriority = THREAD_PRI -ORITY_NORMAL, UINT nStackSize =0, DWORD dwCreateFlags =0, LPSECURITY_ATTRIBUTES lpSecurity -Attrs=NULL);

(2)CwinThread* AfxBeginThread (CruntimeClass* pThreadClass, int nPriority =THREAD_PRIORITY_NOR-MAL, UINT nStackSize =0, DWORD dwCreateFlags =0, LPSECURITY_ATTRIBUTES lpSecurityAttrs=NULL);

形式(1)用于创建工作者线程,形式(2)用于创建用户界面线程。两种形式的返回值相同,都是新创建线程对象的指针。参数意义如下:

pfnThreadProc:工作者线程的函数指针,不可以为空。并且工作者线程的函数必须声明为如下格式:

UINT MyControllingFunction(LPVOID pParam)

pParam:从 CwinThread 类继承来的对象的 RUN-TIME_CLASS 指针。

nPriority:线程的优先级。如果为 0,则与创建它的线程优先级相同。

nStackSize：以字节为单位指定新线程的堆栈大小。如果为0，则与创建它的线程的堆栈大小相同。

dwCreateFlags：指定一个额外的标志控制线程的产生。如果标志是CREATE_SUSPENDED时，以挂起数为1启动线程；如果标志为0，则在创建线程后立即执行线程。

lpSecurityAttrs：指向定义了线程安全属性的SECURITY_ATTRIBUTES结构的指针。如果为NULL，则新线程和创建线程具有同样的安全属性。

线程的终止取决于下列事件之一：线程函数返回、线程调用ExitThread ()、异常情况下用线程的句柄调用TerminateThread()退出、线程所属的进程被终止。

1.2 线程之间的同步

当多个线程同时存在于一个进程中时有可能会同时访问到同一个对象（包括全局变量、系统资源等）。VC++提供了同步对象来协调多线程的并行，常用以下几种方法：

CSemaphore：信号灯对象，允许一定数目的线程访问某个共享资源，常用来控制访问共享资源的线程数量。

CMutex：互斥对象，一个时刻至多只允许一个线程访问某资源，未被占用时处于有信号状态，可以实现对共享资源的互斥访问。

CEvent：事件对象，用于使一个线程通知其他线程某一事件的发生，所以可以用来封锁对某一资源的访问，直到线程释放资源使其成为有信号状态。适用于某一线程等待某事件发生才能执行的场合。

CCriticalSection：临界区对象，将一段代码置入临界区，只允许最多一个线程进入执行这段代码。一个临界区仅在创建它的进程中有效。

1.3 多线程在串口通信中的应用

在创建用来读写串口的线程之前，需要用SetCommMask()函数设置事件掩模来监视指定通信端口上的事件。在用SetCommMask()指定了有用的事件后，可以用WaitCommEvent()函数来等待此事件发生，此函数用在之后建立的线程中。用完SetCommMask()后建立两个线程，一个用来监视串口事件的发生，一个用来读取串口数据。

监视串口事件线程(CommProc)可以监视串口动作，一旦有数据传入就可以让应用程序运行读取串口线程(ReadThread)来读取数据。

事件对象(CEvent)恰能满足串口编程的需要，它是WIN32提供的最灵活的线程间同步方式。用CreateEvent()函数创建两个事件对象：m_hPostMsgEvent和m_hReadSerialEvent。

在CommProc中一直查询串口的状况，具体方法是：判断ComStat.cbInQue的值，一旦不为零就用SetEvent()函数将m_hReadSerialEvent变为有信号状态，同时在ReadThread线程中调用WaitForSingleObject()无限等待至m_hReadSerialEvent变为有信号状态，当其为有信号状态时可运行数据读取与处理工作，并在结束一次数据读取后调用SetEvent ()函数将m_hPostMsgEvent变为有信号状态。同样在CommProc中调用WaitForSingleObject()函数检测m_hPostMsgEvent从而得知数据读取是否结束。

1.4 串口编程中的重叠I/O操作

重叠I/O操作是指应用程序可以在后台读或者写数据，而在前台做其他的事情。VC++对于串口作为文件设备处理，打开并设置好串口后即可利用ReadFile()和WriteFile()进行数据的读写。在读写串口时，可以采取同步执行方式，也可以采取重叠I/O方式。同步执行时，函数直到执行完毕才返回，因而同步执行的其他线程会被阻塞，效率下降；而在重叠方式下，调用的读写函数会立即返回，I/O操作在后台进行，这样线程就可以处理其他事物。

2 ODBC技术的应用

2.1 VC++提供的ODBC数据库类

VC++的MFC基类库定义了几个数据库类。在利用ODBC编程时，经常要使用到CDatabase（数据库类）、CRecordSet(记录集类)和CRecordView(可视记录集类)。

CDatabase类对象提供了对数据源的连接，通过它可以对数据源进行操作。CRecordSet类对象提供了从数据源中提取出的记录集。CRecordSet对象通常用于两种形式：动态行集(dynasets)和快照集(snapshots)。CRecordView类对象能以控件的形式显示数据库记录，这个视图是直接连到CRecordSet对象的表视图。

2.2 利用ODBC编程

应用VC++的AppWizard可以自动生成一个ODBC应用程序框架。在生成的应用程序框架View类中，包含一个指向View类对象的指针m_pSet，目的是在视表单和记录集之间建立联系，使得记录集中的查询结果能够很容易地在视表单上显示出来。

要使程序与数据源建立联系，需要使用CDatabase::OpenEx()或CDatabase::Open()函数来进行初始化。数据库对象必须在使用它构造记录集对象之前初始化。使用ClassWizard来创建CRecordSet类的派生类的对象，并向该对象发送一个指向要使用数据源的指针。然后调用CRecordSet派生类的Open()成员函数打开记录集。CRecordSet派生类中的Requery()、AddNew()、Delete()、Edit()和Update()函数分别可对数据库进行查询、增加、删除、编辑和更新等操作。当完成对数据源的操作后，调用Close()函数关闭记录集和数据源连接。

2.3 利用ODBC实现数据实时处理

因为串口的读取工作是在ReadThread线程中进行的，所以要达到对数据的实时处理应在该线程中利用ODBC对读取的数据进行操作。可以在线程中构造一个CRecordSet派生类的对象，当每次线程利用ReadFile()函数读取完数据并根据相关通信协议判断数据信息无误

后可紧跟其后用 CRecordSet::Open()打开记录集，利用 CRecordSet 派生类的对象对数据库进行数据的增加、修改、删除等操作。

3 应用开发实例

3.1 水处理电压电流检测系统总体结构

在水处理电压电流检测系统中，首先通过上位机的用户界面线程设定串口的基本设置(如波特率、停止位、奇偶校验等)，这些设置要保持与下位机设置一致。用户界面线程应提供开启串口通信的选项以控制工作者线程接收或发送串口数据，线程之间的同步在这一点的实现中显得尤为重要。必须避免多个线程同时访问到同一对象的情况出现，避免方法可参考 2.2 与 2.3 节。工作者线程接收到串口数据之后传递给用户界面线程，后者对数据进行处理显示，有必要时可以将数据结果反馈给工作者线程，工作者线程根据需要调整接发方式。工作者线程在和用户界面线程通信的同时也可通过 ODBC 把数据储存到数据库中。用户界面线程也可以通过 ODBC 直接访问数据查询历史数据。总体结构图如图 1 所示。

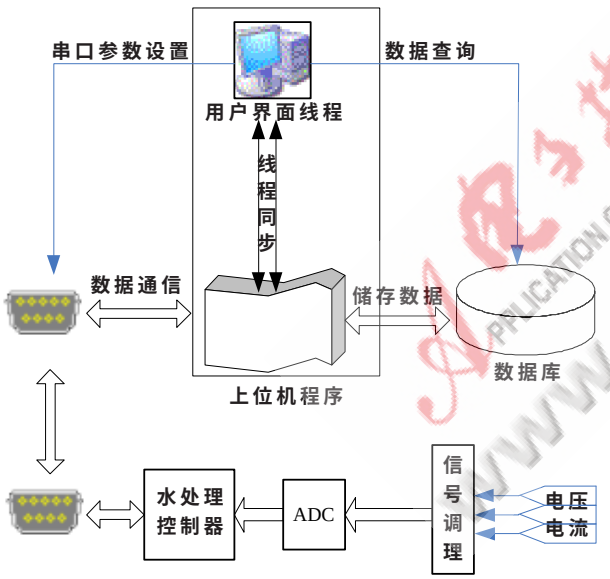

图 1 水处理电压电流检测系统总体结构

3.2 串口监视及数据读取关键代码

以下为两个工作者线程程序，分别实现了串口监视与数据读取。

```
UINT CommProc(LPVOID pParam)        //监视串口字符事件线程
{
                                    …//初始设置略
while(pSerial->m_bConnected)        //串口打开时处于监视状态
{
 ClearCommError (pSerial ->m_hCom,&dwErrorFlags,&ComStat);
 if(ComStat.cbInQue)
 {
                    //通知读串口的线程有串口接收到数据
 SetEvent(pSerial->m_hReadSerialEvent);
   //无限等待读串口数据的线程完成读串口的任务
 WaitForSingleObject(pSerial->m_hPostMsgEvent, INFINITE);
 ResetEvent(pSerial->m_hPostMsgEvent);
 continue;
 }
   dwMask=0;
   if(! WaitCommEvent(pSerial->m_hCom, &dwMask, &os))
                                    //重叠操作
   {
     if(GetLastError()==ERROR_IO_PENDING)
                                    //无限等待重叠操作结果
       GetOverlappedResult (pSerial->m_hCom, &os, &dwTrans, TRUE);
     else
     {
       CloseHandle(os.hEvent);
       return (UINT)-1;
     }
   }
 }
 CloseHandle(os.hEvent);
 return 0;
}
UINT ReadSerialThread(LPVOID pParam)      //读串口数据的线程
{
…                                   //各变量定义及初始化略
while(1)
{
                                    //无限等待串口接受到数据
   WaitForSingleObject(pSerial->m_hReadSerialEvent, INFINITE);
   if(! pSerial->m_bConnected)
   {
                                    //通知监视线程串口关闭
     SetEvent(pSerial->m_hPostMsgEvent);
     return 0L;
   }
   …                                //进行相关数据操作略
   ResetEvent(pSerial->m_hReadSerialEvent);
   SetEvent(pSerial->m_hPostMsgEvent);
}
return 1L;
```

```
}
```

3.3 运用 ODBC 查询数据库的代码

由于数据储存与查询的操作大体类似，本文只列举出查询数据库的代码。代码如下：

```
m_List.DeleteAllItems();//m_List 为 VC 中的 List 控件
gOffset=0;                    //全局变量，数据库查询偏移
CString csSql, str;
CKeyRecord RK;                //定义一个 Crecordset 对象
UpdateData(TRUE);
str=m_Year+"-"+m_Month+"-"+m_Day;
csSql.Format ("SELECT * FROM KeyRecord WHERE 日期 LIKE '%s'", str);
                              //SQL 查询语句
RK.Open(AFX_DB_USE_DEFAULT_TYPE, csSql);
                              //以查询语句打开数据库
if(RK.GetRecordCount()==0)
{                             //判断数据库中是否有数据
    AfxMessageBox("无数据");
    return;
}
RK.MoveFirst();               //从首数据开始查询

do
{
    if(RK.m_PolClose! ="")
    {                         //在 List 控件中显示查询到的数据
    m_List.InsertItem(i, RK.m_Time);
    m_List.SetItemText(i, 1, RK.m_PolClose);
    m_List.SetItemText(i, 2, RK.m_PolOpen);
    m_List.SetItemText(i, 3, RK.m_Date);
    gOffset ++;
    }
    RK.MoveNext();                //移向下一数据
}while(! RK.IsEOF());             //查询终止
RK.Close();                       //关闭数据库
```

将多线程与 ODBC 配合应用于串口编程中，既减少了应用程序对 CPU 的占用时间又实现了串口数据实时地在数据源文件中存储与读取，在水处理电压电流检测系统中运行高效。

参考文献

[1] 求是科技，李现勇.Visual C++串口通信技术与工程实践第二版[M].北京：人民邮电出版社，2004.

[2] 孙鑫，余安萍.VC++深入详解第二版[M].北京：电子工业出版社，2006.

[3] 候俊杰.深入浅出 MFC 第二版[M].武汉：华中科技大学出版社，2001.

[4] BEVERIDGE J，WIENER R.Multithreading Applications in Win 32[M].Addison Wesley Longman，1997.

[5] 乔立岩.多线程测控程序设计方法研究[J].测试技术学报，2002(16)：1145-1149.

[6] 张明慧，张尧禹，黄廉卿.多线程技术在实时测量系统中的应用[J].计算机应用研究，2004(8)：163-165.

(收稿日期：2009-02-11)