# MSComm在三种情况下的使用方法和差别

1 串行通信控件MSComm

2 在单线程中实现自定义的串口通信类

3 多线程下实现串行通信

## 摘要:

本文介绍了在Windows平台下串行通信的实现机制，讨论了根据不同的条件用Visual C++ 设计串行通信程序的三种方法，并结合实际，实现对温度数据的接收监控。

在实验室和工业应用中，串口是常用的计算机与外部串行设备之间的数据传输通道，由于串行通信方便易行，所以应用广泛。依据不同的条件实现对串口的灵活编程控制是我们所需要的。

在光学镜片镀膜工艺中，用单片机进行多路温度数据采集控制，采集结果以串行方式进入主机，每隔10s向主机发送一次采样数据，主机向单片机发送相关的控制命令，实现串行数据接收，处理，记录，显示，实时绘制曲线。串行通信程序开发环境为 VC++ 6.0。

## Windows下串行通信

与以往DOS下串行通信程序不同的是，Windows不提倡应用程序直接控制硬件，而是通过Windows操作系统提供的设备驱动程序来进行数据传递。串行口在Win 32中是作为文件来进行处理的，而不是直接对端口进行操作，对于串行通信，Win 32 提供了相应的文件I/O函数与通信函数，通过了解这些函数的使用，可以编制出符合不同需要的通信程序。与通信设备相关的结构有COMMCONFIG ，COMMPROP，COMMTIMEOUTS，COMSTAT，DCB，MODEMDEVCAPS，MODEMSETTINGS共7个，与通信有关的Windows API函数共有26个，详细说明可参考MSDN帮助文件。以下将结合实例，给出实现串行通信的三种方法。

## 方法一：使用VC++提供的串行通信控件MSComm

首先，在对话框中创建通信控件，若Control工具栏中缺少该控件，可通过菜单Project --> Add to Project --> Components and Control插入即可，再将该控件从工具箱中拉到对话框中。此时，你只需要关心控件提供的对 Windows 通讯驱动程序的 API 函数的接口。换句话说，只需要设置和监视MSComm控件的属性和事件。

在ClassWizard中为新创建的通信控件定义成员对象（CMSComm m\_Serial），通过该对象便可以对串口属性进行设置，MSComm 控件共有27个属性，这里只介绍其中几个常用属性：

CommPort 设置并返回通讯端口号，缺省为COM1。

Settings 以字符串的形式设置并返回波特率、奇偶校验、数据位、停止位。

PortOpen 设置并返回通讯端口的状态，也可以打开和关闭端口。

Input 从接收缓冲区返回和删除字符。

Output 向发送缓冲区写一个字符串。

InputLen 设置每次Input读入的字符个数，缺省值为0，表明读取接收缓冲区中的全部内容。

InBufferCount 返回接收缓冲区中已接收到的字符数，将其置0可以清除接收缓 冲区。

InputMode 定义Input属性获取数据的方式（为0：文本方式；为1：二进制方式）。

RThreshold和SThreshol属性，表示在OnComm事件发生之前，接收缓冲区或发送缓冲区中可以接收的字符数。

以下是通过设置控件属性对串口进行初始化的实例：

BOOL CSampleDlg:: PortOpen()

{

BOOL m\_Opened;

......

m\_Serial.SetCommPort(2); // 指定串口号

m\_Serial.SetSettings("4800,N,8,1"); // 通信参数设置

m\_Serial.SetInBufferSize(1024); // 指定接收缓冲区大小

m\_Serial.SetInBufferCount(0); // 清空接收缓冲区

m\_Serial.InputMode(1); // 设置数据获取方式

m\_Serial.SetInputLen(0); // 设置读取方式

m\_Opened=m\_Serail.SetPortOpen(1); // 打开指定的串口

return m\_Opened;

}

打开所需串口后，需要考虑串口通信的时机。在接收或发送数据过程中，可能需要监视并响应一些事件和错误，所以事件驱动是处理串行端口交互作用的一种非常有效的方法。使用 OnComm 事件和 CommEvent 属性捕捉并检查通讯事件和错误的值。发生通讯事件或错误时，将触发 OnComm 事件，CommEvent 属性的值将被改变，应用程序检查 CommEvent 属性值并作出相应的反应。在程序中用ClassWizard为CMSComm控件添加OnComm消息处理函数：

void CSampleDlg::OnComm()

{

......

switch(m\_Serial.GetCommEvent())

{

case 2:

// 串行口数据接收，处理；

}

}

## 方法二：在单线程中实现自定义的串口通信类

控件简单易用，但由于必须拿到对话框中使用，在一些需要在线程中实现通信的应用场合，控件的使用显得捉襟见肘。此时，若能够按不同需要定制灵活的串口通信类将弥补控件的不足，以下将介绍如何在单线程中建立自定义的通信类。该通信类CSimpleComm需手动加入头文件与源文件，其基类为CObject，大致建立步骤如下：

(1) 打开串口，获取串口资源句柄

通信程序从CreateFile处指定串口设备及相关的操作属性。再返回一个句柄，该句柄将被用于后续的通信操作，并贯穿整个通信过程。

CreateFile()函数中有几个值得注意的参数设置：串口共享方式应设为0，串口为不可共享设备；创建方式必须为OPEN\_EXISTING,即打开已有的串口。对于dwFlagAndAttribute参数，对串口有意义的值是FILE\_FLAG\_OVERLAPPED，该标志表明串口采用异步通信模式，可进行重叠操作；若值为NULL，则为同步通信方式，在同步方式下，应用程序将始终控制程序流，直到程序结束，若遭遇通信故障等因素，将导致应用程序的永久等待，所以一般多采用异步通信。

(2)串口设置

串口打开后，其属性被设置为默认值，根据具体需要，通过调用GetCommState(hComm,&dcb)读取当前串口设备控制块DCB（Device Control Block）设置，修改后通过SetCommState(hComm,&dcb)将其写入。再需注意异步读写的超时控制设置，通过COMMTIMEOUTS结构设置超时，调用SetCommTimeouts(hComm,&timeouts)将结果写入。以下是温度监控程序中串口初始化成员函数：

BOOL CSimpleComm::Open( )

{

DCB dcb;

m\_hIDComDev=CreateFile( "COM2", GENERIC\_READ | GENERIC\_WRITE,0,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL|FILE\_FLAG\_OVE RLAPPED, NULL ); // 打开串口，异步操作

if( m\_hIDComDev == NULL ) return( FALSE );

dcb.DCBlength = sizeof( DCB );

GetCommState( m\_hIDComDev, &dcb ); // 获得端口默认设置

dcb.BaudRate=CBR\_4800;

dcb.ByteSize=8;

dcb.Parity= NOPARITY;

dcb.StopBits=(BYTE) ONESTOPBIT;

...... }

（3）串口读写操作

主要运用ReadFile（）与WriteFile（）API函数，若为异步通信方式，两函数中最后一个参数为指向OVERLAPPED结构的非空指针，在读写函数返回值为FALSE的情况下，调用GetLastError（）函数，返回值为ERROR\_IO\_PENDING，表明I/O操作悬挂，即操作转入后台继续执行。此时，可以用WaitForSingleObject()来等待结束信号并设置最长等待时间，举例如下：

BOOL bReadStatus;

bReadStatus = ReadFile( m\_hIDComDev, buffer,

dwBytesRead, &dwBytesRead, &m\_OverlappedRead );

if(!bReadStatus)

{

if(GetLastError()==ERROR\_IO\_PENDING)

{

WaitForSingleObject(m\_OverlappedRead.hEvent,1000);

return ((int)dwBytesRead);

}

return(0);

}

return ((int)dwBytesRead);

定义全局变量m\_Serial作为新建通信类CSimpleComm的对象，通过调用类的成员函数即可实现所需串行通信功能。

与方法一相比，方法二赋予串行通信程序设计较大的灵活性，端口的读写可选择较简单的查询式，或通过设置与外设数据发送时间间隔TimeCycle相同的定时器：SetTimer(1,TimeCycle,NULL)，进行定时读取或发送。

CSampleView:: OnTimer(UINT nIDEvent)

{

char InputData[30];

m\_Serial.ReadData(InputData,30);

// 数据处理

}

若对端口数据的响应时间要求较严格，可采用事件驱动I/O读写，Windows定义了9种串口通信事件，较常用的有：

EV\_RXCHAR: 接收到一个字节，并放入输入缓冲区。

EV\_TXEMPTY: 输出缓冲区中的最后一个字符发送出去。

EV\_RXFLAG: 接收到事件字符(DCB结构中EvtChar成员)，放入输入缓冲区。

在用SetCommMask()指定了有用的事件后，应用程序可调用WaitCommEvent()来等待事件的发生。

SetCommMask(hComm,0)可使WaitCommEvent()中止。

## 方法三 多线程下实现串行通信

方法一，二适用于单线程通信。在很多工业控制系统中，常通过扩展串口连接多个外设，各外设发送数据的重复频率不同，要求后台实时无差错捕捉，采集，处理，记录各端口数据，这就需要在自定义的串行通信类中创建端口监视线程，以便在指定的事件发生时向相关的窗口发送通知消息。

线程的基本概念可详见VC++参考书目，Windows内部的抢先调度程序在活动的线程之间分配CPU时间，Win 32 区分两种不同类型的线程，一种是用户界面线程UI（User Interface Thread）,它包含消息循环或消息泵，用于处理接收到的消息；另一种是工作线程（Work Thread），它没有消息循环，用于执行后台任务。用于监视串口事件的线程即为工作线程。

多线程通信类的编写在端口的配置，连接部分与单线程通信类相同，在端口配置完毕后，最重要的是根据实际情况，建立多线程之间的同步对象，如信号灯，临界区，事件等，相关细节可参考VC++ 中的同步类。一切就绪后即可启动工作线程：

CWinThrea \*CommThread = AfxBegin

Thread(CommWatchThread, // 线程函数名

(LPVOID) m\_pTTYInfo, // 传递的参数

THREAD\_PRIORITY\_ABOVE\_NORMAL, // 设置线程优先级

(UINT) 0, // 最大堆栈大小

(DWORD) CREATE\_SUSPENDED , // 创建标志

(LPSECURITY\_ATTRIBUTES) NULL); // 安全性标志

　　同时，在串口事件监视线程中：

if(WaitCommEvent(pTTYInfo->idComDev,&dwEvtMask,NULL))

{

if((dwEvtMask & pTTYInfo->dwEvtMask )== pTTYInfo->dwEvtMask)

{

WaitForSingleObject(pTTYInfo->hPostEvent,0xFFFFFFFF);

ResetEvent(pTTYInfo->hPostEvent); // 置同步事件对象为非信号态

::PostMessage(CSampleView,ID\_COM1\_DATA,0,0); // 发送通知消息

}

}

用PostMessage()向指定窗口的消息队列发送通知消息，相应地，需要在该窗口建立消息与成员函数间的映射，用ON\_MESSAGE将消息与成员函数名关联。

BEGIN\_MESSAGE\_MAP(CSampleView, CView)

//{{AFX\_MSG\_MAP(CSampleView)

ON\_MESSAGE(ID\_COM1\_DATA, OnProcessCom1Data)

ON\_MESSAGE(ID\_COM2\_DATA, OnProcessCom2Data)

.....

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

然后在各成员函数中完成对各串口数据的接收处理，但必须保证在下一次监测到有数据到来之前，能够完成所有的中间处理工作。否则将造成数据的捕捉错误。

多线程的实现可以使得各端口独立，准确地实现串行通信，使串口通信具有更广泛的灵活性与严格性，且充分利用了CPU时间。但在具体的实时监控系统中如何协调多个线程，线程之间以何种方式实现同步也是在多线程串行通信程序实现的难点。

以VC++ 6.0 为工具，实现串行通信的三种方法各有利弊， 根据不同需要，选择合适的方法，将达到事半功倍的效果。在温度监控系统中，笔者采用了方法二，在Window 98 ,Windows 95 上运行稳定，取得了良好的效果。

end~!